**Universidade de Aveiro**
**2013**
Departamento de Electrónica, Telecomunicações e Informática

**Gonçalo José Correia Castanheiro**

# Controlo e navegação em robótica móvel com base em Arduino e ZigBee

**Universidade de Aveiro**
**2013**
Departamento de Electrónica, Telecomunicações e Informática

**Gonçalo José Correia Castanheiro**

# Controlo e navegação em robótica móvel com base em Arduino e ZigBee

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Dr. António Guilherme Rocha Campos, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Dr. Francisco José Curado Mendes Teixeira, Investigador do Centro de Estudos do Ambiente e do Mar (CESAM) da Universidade de Aveiro.

*“If I have seen further it is by standing on the shoulders of giants.”*

- Sir Isaac Newton

**o júri**

presidente

Professor Doutor Manuel Bernardo Salvador Cunha
Professor Auxiliar, Universidade de Aveiro

vogais

Professor Doutor Armando Jorge Miranda de Sousa
Professor Auxiliar, Universidade do Porto - Faculdade de Engenharia

Professor Doutor António Guilherme Rocha Campos
Professor Auxiliar, Universidade de Aveiro

**agradecimentos**

Quero agradecer aos meus pais por todo o apoio, esforço e carinho ao longo de toda a minha vida. Sem eles, isto não passaria de um sonho.
Gostava também de deixar uma palavra de gratidão aos meus amigos dentro e fora do seio académico, por terem ajudado a superar os momentos de maior desespero.
Agradeço profundamente ao meu orientador, o professor Guilherme Campos pela constante motivação e palavras de coragem que foram determinantes em várias etapas.
Quero deixar um agradecimento especial ao meu orientador, o professor Francisco Curado, pelo conhecimento transmitido e, mais importante, guiar-me na direcção certa para a aprendizagem.
Gostaria também de agradecer ao João Quintas pela ajuda em várias etapas do trabalho. Contribuiu largamente para a evolução do trabalho, bem como na fase de testes. Sem ele, a conclusão desta dissertação seria muito mais complicada.

**palavras-chave**

robótica móvel, navegação cooperativa, controlo de motores, programação de *robots*, programação por comportamentos, fusão sensorial, filtros de partículas

**resumo**

Esta dissertação aborda o tema da navegação cooperativa em robótica móvel. Dentro deste vasto assunto, dá-se especial ênfase à comunicação entre os vários veículos robóticos e entre estes e o nó de processamento central (PC) onde é executado o algoritmo de navegação cooperativa. São apresentadas soluções alternativas em termos de protocolo de comunicações, tendo em vista os objectivos do presente trabalho.

O projecto envolveu o desenvolvimento de um veículo robótico autónomo munido de sensores de distância (infravermelhos e ultra-sons), bússola digital, sensores de odometria (*encoders*) e actuadores (motores DC e servomotor), todos eles ligados a um microcontrolador *Arduino*. É adoptado o protocolo *ZigBee* para a comunicação entre todos os nós do sistema cooperativo, incluindo o nó de processamento central. A construção e teste do veículo é descrita e documentada em detalhe (incluindo registos fotográficos).

Exploram-se os tópicos da percepção, localização e actuação em robots móveis, e propõem-se soluções para controlo de motores e seguimento de trajectórias.

O trabalho envolve o estudo de diversos sensores e motores, bem como arquitecturas para a programação de *robots*, nomeadamente a arquitectura baseada em comportamentos hierarquizados em níveis de prioridade. Neste contexto, são apresentados e descritos detalhadamente os comportamentos implementados.

Foram realizados testes parcelares, nomeadamente de caracterização e calibração dos sensores, configuração e desempenho das comunicações e afinação do controlo das rodas motrizes, seguidos de teste de funcionamento global. Para este efeito, definiu-se uma tarefa de aquisição de dados sensoriais ao longo de um percurso com etapas previamente definidas. Esses dados são transmitidos ao PC e utilizados por um algoritmo de localização 2D baseado num filtro de partículas. Os resultados deste teste são analisados em detalhe.

São discutidos possíveis caminhos de evolução em trabalho futuro.

**keywords**

mobile robotics, cooperative navigation, motor control, robot programming, behaviors based programming, sensor fusion, particle filters

**abstract**

This dissertation addresses the topic of cooperative navigation in mobile robotics. Within this vast subject, particular emphasis is placed on the communication among the various robotic vehicles and between these and the central processing node (PC) executing the cooperative navigation algorithm. Alternative communication protocols are presented in the context of the envisaged implementation.

An autonomous robotic vehicle was developed in the course of the project, equipped with range sensors (infra-red and ultra-sonic), a digital compass, odometers (encoders) and actuators (DC and servo motors), all connected to an *Arduino* micro-controller. The *ZigBee* protocol is adopted for communication among all the nodes in the cooperative system, including the central processing node. The construction and testing of this vehicle is described and documented in detail (including photos).

The work explores the topics of perception, localisation and actuation in mobile robots and proposes solutions for motor control and path following.

Various sensors and actuators are studied, as well as robot programming architectures, especially behaviour-based programming using hierarchical priority levels. The behaviours actually implemented on the robotic vehicle are presented and described in detail.

Following partial tests for sensor characterisation and calibration, configuration and performance assessment of the communication links and wheel motor drive adjustment, a global operation test was carried out. For this purpose, a sensor data acquisition task was defined over a pre-defined path comprising several different stages. The data are transmitted to the PC to feed a 2D localization algorithm based on a particle filter. The test results are analysed in detail.

Possible future work threads are discussed.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Glossário

ADC – *Analogic-Digital Converter*
CSMA-CA – *Carrier Sense Multiple Access with Collision Avoidance*
DSSS – *Direct Sequence Spread Spectrum*
FFD – *Full-Function Device*
FP – Filtro de Partículas
GUI – *Graphical User Interface*
IEEE – *Institute of Electrical and Electronics Engineers*
IR – *Infrared Distance Sensor*
ISM – *Industrial, Science and Medical*
MAC – *Medium Access Control*
MC – *Monte Carlo*
PAN – *Personal Area Network*
PWM – *Pulse Width Modulation*
RF – *Radio Frequency*
RFD – *Reduced-Function Device*
ROS – *Robot Operating System*
SCL – *Serial Clock*
SDA – *Serial Data*
SLAM – *Simultaneous Localization and Mapping*
*Swarm Robotics* – traduzido do inglês como "*enxame de robots*", e trata-se de um sistema robótico constituído por diversos *robots* (tipicamente dezenas ou centenas) muito básicos que comunicam entre si para atingirem um objectivo
UART – *Universal Asynchronous Receiver/Transmitter*
US – *Ultrasonic Range Finder*
WLAN – *Wireless Local Area Network*
WPAN – *Wireless Personal Area Network*
WSN – *Wireless Sensor Network*

# 1. Introdução

Esta dissertação insere-se na área da robótica móvel cooperativa.

A navegação de uma equipa de agentes robóticos móveis pode ser útil em variados cenários, nomeadamente em ambientes de difícil acesso (ex: subaquáticos de elevada profundidade, escombros) ou potencialmente prejudiciais para o ser humano (ex: áreas minadas ou radioactivas). O desenvolvimento desta área tem ganho visibilidade sobretudo através de competições como o futebol robótico.

A cooperação entre os veículos robóticos autónomos implica que estes sejam capazes de trocar informação entre si, incluindo dados sensoriais e eventualmente a estimativa actualizada da sua localização. No cenário que se pretende explorar, além de comunicarem entre si, devem também fazê-lo com uma estação central de processamento (PC), para que esta receba os dados sensoriais e execute o algoritmo de navegação cooperativa. A situação é ilustrada na Figura 1.1.

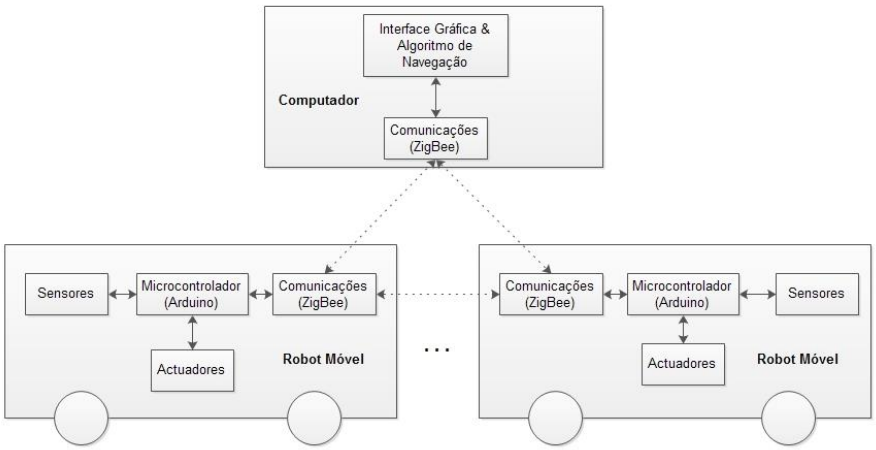



**Figura 1.1** – Estrutura de um sistema robótico cooperativo.

Os objectivos principais deste projecto consistem então na implementação de uma rede de comunicação *wireless* para trocas de informação entre um PC e um ou mais robots móveis e na configuração do seu sistema de sensores e actuadores. Usando a rede de comunicações, o PC deve conseguir receber informação sensorial proveniente dos veículos robóticos, que serão equipados com microcontroladores *Arduino*. Com base nesta informação, deve estimar a localização de cada num mapa previamente conhecido e enviar novas instruções de comando. O protocolo de comunicações usado será o *ZigBee*; é necessário conectar um módulo *XBee* devidamente configurado a cada elemento constituinte da rede.

Destacam-se assim claramente três áreas principais de estudo nesta dissertação:

- Comunicações Sem Fios;
- Sensores e Actuadores;
- Integração no Sistema de Navegação Cooperativa.

Para testar os conceitos abordados neste trabalho e conseguir desenvolver um protótipo em tempo útil e a custos reduzidos, decidiu-se utilizar, tanto quanto possível, *software* e *hardware open source*. Esta opção permite aceder sem custos a um vasto leque de documentação e *software* já desenvolvido e testado. A escolha do microcontrolador Arduino enquadra-se nesta filosofia. Trata-se de um dispositivo muito popularizado por ser versátil e fácil de programar, estando disponíveis muitos exemplos práticos de aplicação em diversas áreas e fóruns de troca de informação.

Foi construído um veículo robótico equipado com actuadores que lhe permitem movimentar-se e ajustar-se ao meio em que se movimenta, vários sensores para percepcionar esse meio e a situação do veículo relativamente a ele e um módulo *XBee* para comunicar com outros kits robóticos e com o PC (também ele munido de um módulo *XBee*). Os sensores utilizados são: sensor de distância por luz infravermelha, sensor de distância por ultra-sons, bússola digital, e *encoders* acoplados a cada uma das rodas motrizes. Os actuadores são motores DC acoplados a essas rodas e um motor servo destinado a orientar os sensores de distância segundo o ângulo desejado. Todo este equipamento, bem como o módulo de comunicações sem fios *XBee*, é ligado ao microcontrolador *Arduino*. Este adquire e regista os dados sensoriais, efectua a sua pré-filtragem e prepara os pacotes de dados para enviar para o PC através do módulo de comunicação *XBee*. A pré-filtragem dos dados sensoriais (por exemplo uma média), cujo objectivo é atenuar os erros de medida, é feita no *Arduino* porque é uma tarefa pouco exigente em termos de processamento, e reduz a quantidade de dados enviada para processamento externo, evitando sobrecarregar o sistema de comunicações.

No PC, o único *hardware* extra utilizado é o módulo *XBee* para comunicar com os veículos móveis. Em termos de *software*, foi desenvolvida uma interface gráfica (GUI - *Graphical User Interface*) em *Matlab* para receber, guardar e representar os dados sensoriais à medida que vão sendo recebidos no PC, bem como para enviar de volta instruções de comando ao veículo robótico. O PC executa um algoritmo de navegação cooperativa, também implementado em *Matlab*, que utiliza os dados definidos pelo utilizador através da GUI e traça os trajectos e localizações estimadas para o veículo robótico. Neste trabalho, a localização do *robot* é feita *offline*, mas o objectivo no futuro é que seja concretizada em tempo-real.

A estrutura da dissertação é a seguinte:

No **capítulo 2** são abordadas as Comunicações Sem Fios. Aqui são introduzidas as comunicações de redes do tipo WPAN (*Wireless Personal Area Network*) e é apresentado o protocolo de comunicação *ZigBee*. São expostas as principais características e especificações técnicas deste protocolo, bem como as vantagens que aconselharam a sua utilização. São apresentados os módulos *XBee* utilizados, os testes de comunicação realizados em vários cenários, e discutidos os respectivos resultados.

O **capítulo 3** trata dos Sensores e Actuadores utilizados no protótipo construído. É apresentada a função de cada um no veículo robótico, o seu modo de funcionamento e, no caso dos sensores de distância, técnicas adoptadas para melhorar o desempenho. São

descritos os testes efectuados aos sensores e actuadores e analisados os resultados obtidos. É introduzido o tema da Fusão Sensorial e explicado o controlo dos motores DC que accionam as rodas motrizes.

O **capítulo 4** discute o *software* de Navegação e Controlo. Após uma introdução sobre arquitectura de veículos robóticos móveis, é apresentada a que foi adoptada neste caso, descrevendo-se em detalhe os comportamentos implementados.

No **capítulo 5** é finalmente apresentada a integração das três componentes objecto dos capítulos anteriores no protótipo construído. É explicado em maior detalhe o funcionamento do microcontrolador (*Arduino*) e a sua integração com o restante *hardware*, bem como as suas tarefas. É abordada a integração do veículo robótico autónomo no sistema de navegação cooperativa (PC), bem como o estabelecimento de comunicações entre ambos. Estuda-se mais a fundo o controlo de motores em trajectórias definidas em ambientes conhecidos.

O **capítulo 6** é dedicado a um teste final global demonstrativo dos resultados obtidos. Nesse teste, fez-se o kit robótico percorrer uma trajectória pré-definida num mapa e transmitir ao PC os dados sensoriais adquiridos para esse efeito. Estes dados são utilizados pelo algoritmo de navegação para estimar a localização do veículo. Este teste permite analisar os principais desafios e expor as dificuldades encontradas. É feita uma avaliação de todo o projecto.

O **7º e último capítulo** apresenta as considerações finais, indicando possíveis pistas para trabalho futuro no sentido de valorizar mais o projecto e expandir o seu potencial de aplicação.

# 2. Comunicações

## 2.1. Introdução

Comunicação é transmissão de informação e pode estabelecer-se não apenas directamente entre seres humanos mas também entre máquinas (computadores). Uma *rede de computadores* estabelece comunicação entre múltiplas máquinas e permite assim a troca de dados entre sistemas e programas (e, em última análise, entre os seus utilizadores humanos). Os computadores (ou outros elementos com capacidade de processamento) que originam, encaminham ou recebem os dados são designados *nós* da rede.

Na década de 60, os cientistas ainda não estavam certos da melhor maneira de interligar dois computadores; interligar milhões, como acontece hoje em dia, com o advento da *Internet*, era uma ideia demasiado rebuscada para sequer imaginar. No entanto, isso tornou-se não só possível como realizável de inúmeras maneiras, em termos do meio físico de transmissão da informação (canal de comunicação) e do modo como ela é codificada (protocolo de comunicação).

Quanto ao canal de comunicação, podem distinguir-se essencialmente dois tipos:

- Com fios (comunicações por cabo de cobre ou fibra óptica);
- Sem fios ou *wireless* (comunicações via rádio, entre outras).

É muito comum o percurso da informação envolver os dois (canal misto). Por exemplo, em comunicações móveis, os telemóveis comunicam com estações locais por rádio, mas a interligação entre estas recorre normalmente a uma infraestrutura cablada em fibra óptica. Em comunicações telefónicas de longa distância pela rede fixa, pode observar-se o inverso, com ligação cablada (em cobre, por exemplo) dos pontos terminais às centrais locais e via satélite entre centrais.

Os protocolos de comunicação definem as regras e os formatos dos dados para as trocas de informação numa dada rede, necessariamente conhecidos e respeitados pelo emissor e pelo receptor. Tem-se desenvolvido uma grande multiplicidade de protocolos, para responder de forma optimizada aos requisitos específicos de cada rede em termos de dimensão, topologia, canal, alcance, autonomia, consumo energético, largura de banda, velocidade de transmissão de dados, fiabilidade, segurança, gestão de prioridades, coexistência com outras redes, etc.

## 2.2. Comunicações em robótica móvel

Em aplicações de robótica móvel, interessam especialmente os protocolos de comunicação sem fios; os mais comuns são:

- *Wi-Fi*
- *Bluetooth*
- *ZigBee*

A **Tabela 2.1** resume e compara as características destes protocolos. É adaptada de **[1]**, onde pode encontrar-se uma análise comparativa aprofundada.

**Tabela 2.1** - Resumo comparativo dos protocolos mais comuns em robótica móvel.

| **Standard** | ***Wi-Fi*** | ***Bluetooth*** | ***ZigBee*** |
|---|---|---|---|
| Norma | IEEE 802.11a/b/g | IEEE 802.15.1 | IEEE 802.15.4 |
| Banda de Frequência | 2.4GHz; 5GHz | 2.4GHz | 868/915MHz; 2.4GHz |
| Máx. velocidade de transmissão | 54Mb/s | 1Mb/s | 250Kb/s |
| Alcance | 100m | 10m | 10 – 100m |
| Potência de Transmissão | 15 – 20dBm | 0 – 10dBm | (-25) – 0dBm |
| Número de canais RF | 14 (2.4GHz) | 79 | 1/10; 16 |
| Largura de Banda | 22MHz | 1MHz | 0.3/0.6MHz; 2MHz |
| Modulação | BPSK, QPSK, COFDM, DDK, M-QAM | GFSK | BPSK (+ASK), O-QPSK |
| *Spreading* | DSSS, CCK, OFDM | FHSS | DSSS |
| Mecanismo de Coexistência | Dynamic freq. selection, transmit power control (802.11h) | Adaptative freq. hopping | Dynamic freq. selection |
| Célula Base | BSS | Piconet | Star |
| Extensão Topológica | ESS | Scatternet | Cluster Tree, Mesh |
| Nº máximo de nós | 2007 | 8 | > 65000 |
| Encriptação | RC4 stream cypher (WEP), AES block cypher | E0 stream cipher | AES block cypher (CTR, counter mode) |
| Autenticação | WPA2 (802.11i) | Shared secret | CBC-MAC (ext. of CCM) |
| Protecção de dados | 32-bit CRC | 16-bit CRC | 16-bit CRC |

Acrónimos:
ASK (*amplitude shift keying*), GFSK (*Gaussian frequency SK*), BPSK/QPSK (*binary/quadrature phase SK*), O-QPSK (*offset-QPSK*), OFDM (*orthogonal frequency division multiplexing*), COFDM (*coded OFDM*), MB-OFDM (*multiband OFDM*), M-QAM (*M-ary quadrature amplitude modulation*), CCK (*complementary code keying*), FHSS/DSSS (*frequency hopping/direct sequence spread spectrum*), BSS/ESS (*basic/extended service set*), AES (*advanced encryption standard*), WEP (*wired equivalent privacy*), WPA (*Wi-Fi protected access*), CBC-MAC (*cipher block chaining message authentication code*), CCM (*CTR with CBC-MAC*), CRC (*cyclic redundancy check*).

A **Figura 2.1**, adaptada de **[1]**, apresenta uma comparação entre os consumos energéticos dos protocolos em questão.

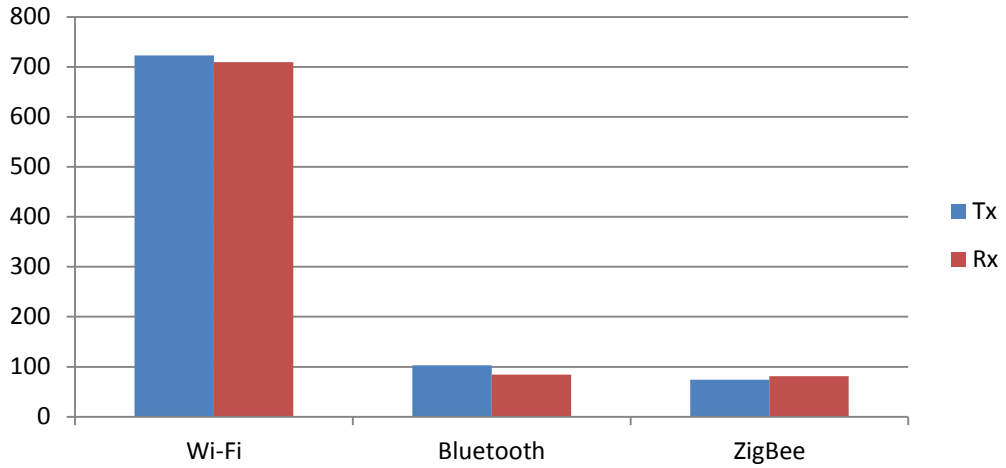



**Figura 2.1** - Consumos energéticos dos protocolos em análise.

O primeiro protocolo (*Wi-Fi*), regido pela norma IEEE 802.11, enquadra-se nas redes de comunicações da classe WLAN (*Wireless Local Area Network*). Permite troca de dados entre dispositivos, ou ligação à Internet, com alcance e taxa de transmissão relativamente elevados, o que implica um também elevado consumo energético. Não é, por isso, a melhor solução quando se pretende um sistema alimentado por baterias com grande autonomia. Podem encontrar-se muitas aplicações de redes *Wi-Fi* em robótica móvel.

Em **[2]**, o *Wi-Fi* é usado num cenário de localização e mapeamento simultâneos (SLAM – *Simultaneous Localization and Mapping*) com um *robot* equipado com uma camara de vídeo de profundidade (*Kinect*).

**[3]** refere-se a uma formação de veículos robóticos interligados por *Wi-Fi*, em que a potência do sinal recebido de um *router* é usada por veículo para estimar a sua posição relativa.

Em **[4]** é desenvolvida uma tese de mestrado em volta da utilização de uma rede *ZigBee* para localização de um veículo robótico em ambiente *indoor* com base em triangulação. É ainda feito um estudo comparativo entre *ZigBee*, *Bluetooth* e *Wi-Fi*.

Os protocolos *Bluetooth* e *ZigBee* enquadram-se numa classe de redes sem fios designada WPAN. Este tipo de redes é pensado para cobrir a vizinhança próxima de uma pessoa ou objecto; tipicamente, o seu alcance estende-se radialmente até 10m. O comité formado para normalização destas redes é o IEEE 802.15. O seu trabalho focou-se nos ideais de baixo custo, baixo consumo energético e atravancamento reduzido. Sob este comité, funcionam grupos que exploram diferentes enquadramentos de aplicação de WPAN e definem normas para cada um, nomeadamente o IEEE 802.15.1, que enquadra o protocolo *Bluetooth* e o IEEE 802.15.4 que enquadra (entre outros) o protocolo *ZigBee*.

O *Bluetooth*, que opera na banda de frequência dos 2.4GHz, ganhou grande popularidade para interligar dispositivos como telemóveis, computadores, câmaras digitais ou impressoras. É tipicamente utilizado para transferir dados áudio (por exemplo entre

telemóvel e auricular sem fios) e pequenos ficheiros ente telemóveis ou ente telemóveis e computadores.

Em **[5]** , o *Bluetooth* é usado para detecção de proximidade de outros dispositivos, no contexto de comunicação entre veículos robóticos reactivos. Neste cenário, um *robot* fica "preso" e incapaz de se soltar; perante esta situação, o *robot* deve ser capaz de "avisar" outros veículos robóticos acerca do seu problema, de forma a evitar que outros venham a sofrer do mesmo problema.

Em **[6]** é descrito um trabalho em que o *Bluetooth*, bem como uns dispositivos denominados de Sun SPOT's são utilizados para controlo remoto de um veículo robótico através de uma plataforma (*browser*) na Internet. Os mencionados Sun SPOT's são dispositivos com capacidade de processamento, sensores integrados e são capazes de comunicar sem fios entre si. Curiosamente, o protocolo de comunicação utilizado por estes é baseado na norma 802.15.4.

O *ZigBee* **[7]** será apresentado em detalhe em subsecção própria (**2.4**), por ser o protocolo escolhido neste trabalho. Apresenta como principal vantagem o seu baixo consumo energético. A título de exemplos de aplicação na literatura, refiram-se desde já o estudo de desempenho num sistema de *swarm robotics* apresentado em **[8]** e o cálculo de um mapa probabilístico de localização **[9]** em que foi usada a potência do sinal recebido para estimar as distâncias entre os nós da rede.

## 2.3. Redes WSN

É interessante, neste ponto, mencionar as *redes de sensores sem fios* (WSN – *Wireless Sensor Networks*), de uso crescente em robótica, sobretudo para efeitos de monitorização de características físicas ou ambientais. As WSN são constituídas por nós, cada um conectado a um sensor autónomo, distribuídos por uma área a monitorizar. Tipicamente, cada nó/sensor integra três elementos principais: um rádio emissor/receptor, um microcontrolador e uma fonte de energia (normalmente baterias) **[10] [11]**. Trabalham cooperativamente, reencaminhando os dados adquiridos até um dispositivo denominado *gateway*, como ilustra a Figura 2.2. A *gateway* direcciona os dados para um computador ou outra rede externa.


**Figura 2.2** - Rede WSN **[10]**.

As redes de sensores têm grande aplicabilidade em robótica móvel, podendo auxiliar em planeamento de trajectos, localização e coordenação entre múltiplos *robots*. Em contrapartida, o seu desempenho pode ser optimizado (nomeadamente em termos de

distribuição dos nós, transporte e agregação de dados e reacção à falha de sensores) com o auxílio de meios robóticos. Por exemplo, os efeitos da mobilidade numa rede de sensores podem ser testados instalando nós em *robots* móveis. Em **[12]**, é feito um levantamento de uma série de aplicações de robótica em redes de sensores *wireless*.

Naturalmente, os protocolos usados nestas redes dependem dos requisitos da aplicação, mas são muito frequentemente baseados nas normas IEEE 802.15.4 e IEEE 802.11. Tipicamente, exigem alimentação por baterias, o que torna vantajoso o recurso ao *ZigBee*; noutros casos, dá-se prioridade à largura de banda, o que favorece a opção por *Wi-Fi* **[10]**.

## 2.4. Protocolo *ZigBee*

### 2.4.1. Caracterização geral

Este protocolo enquadra-se, como já foi referido, na norma IEEE 802.15.4. Esta define uma solução de baixa taxa de transmissão de dados, elevada autonomia energética e implementação/utilização pouco complexa; visa aplicações como sensores, brinquedos interactivos, telecomandos e domótica. As principais características das camadas física e MAC (*Medium Access Control*) são:

- Taxas de transmissão de 250kbps, 40kbps e 20kbps;
- Dois modos de endereçamento; endereços de 16bit e 64bit;
- Suporte para dispositivos de latência crítica, como *joysticks*;
- Algoritmo de acesso ao meio CSMA-CA;
- Criação da rede automática pelo coordenador;
- Protocolo completamente *handshake* para fiabilidade de transferência;
- Gestão de potência para garantir o baixo consumo de energia;
- 16 canais na banda 2.4GHz ISM, 10 canais na banda 915MHz I e um canal na banda 868MHz.

O protocolo *ZigBee* e a família de normas de comunicação que têm sido desenvolvidas no seu âmbito são geridas por uma organização sem fins lucrativos designada *ZigBee Alliance*, criada em 2002 e formada por promotores e participantes. Os promotores são os elementos de maior influência e entre eles contam-se empresas como a *Philips*, a *Texas Instruments* e a *Silicon Labs*. Entre os participantes, que também desempenham um papel activo na evolução do *ZigBee*, embora não tão determinante, podem destacar-se nomes como *AT&T*, *Cisco* ou *Samsung* **[13]**.

O IEEE e a *ZigBee Alliance* trabalham em conjunto por forma a especificarem toda a *stack* protocolar. O grupo IEEE 802.15.4 foca-se na especificação das duas camadas mais baixas (física e MAC), enquanto a *ZigBee Alliance* procura definir as camadas superiores (camada de rede à camada de aplicação). Um dos objectivos principais é garantir a interoperabilidade das redes, de forma a que os consumidores possam adquirir produtos de diferentes fabricantes com a confiança de que estes funcionam em conjunto. A norma vem-se estendendo às mais diversas áreas, desde a saúde às aplicações de domótica.

O nome *ZigBee* deve é inspirado no ziguezaguear das abelhas entre as flores, enquanto transmitem informação às outras abelhas sobre os locais onde podem encontrar mais recursos. É especialmente direccionado para aplicações de automação e controlo remoto. Estabelece um compromisso entre a taxa de transmissão de dados (menor do que no caso do *Bluetooth*) e a duração das baterias dos dispositivos ligados à rede (tipicamente muito superior). O *ZigBee* pode também implementar redes *mesh* maiores do que o *Bluetooth* **[13, 14]**.

Os dispositivos sem fios compatíveis com *ZigBee* devem consiguir transmitir dados até 10-75 metros, dependendo do ambiente RF e do consumo energético necessário para uma dada aplicação. Devem também ser capazes de operar na banda RF não licenciada (2.4GHz a nível mundial, 915MHz para as Américas e 868MHz para a Europa). As taxas de transmissão de dados possíveis são:

- 250kbps na banda de 2.4GHz
- 40kbps na banda de 915MHz
- 20kbps na banda de 868MHz

### 2.4.2. Componentes de uma WPAN – *ZigBee*

A norma IEEE 802.15.4 define dois tipos de componentes de uma WPAN: FFD (*Full-Function Device*) e RFD (*Reduced-Function Device*). Deve haver pelo menos um FFD, operando como o coordenador da rede.

Um FFD pode operar em três modos: *coordenador da PAN*, *coordenador* ou *dispositivo terminal*. Por sua vez, um RFD é direccionado para aplicações muito simples, que não envolvem o envio de grandes quantidades de dados. Um FFD pode comunicar com componentes RFD ou FFD, ao passo que um RFD pode apenas comunicar com um FFD.

Ao nível da camada de rede do *ZigBee*, aplicam-se outras designações: ***ZigBee end-device*** corresponde a um RFD ou a um FFD operando como um dispositivo terminal; ***ZigBee router*** é um FFD com capacidades de *routing*; ***ZigBee coordinator*** (único na rede) é o FFD coordenador da PAN.

A **Figura 2.3** ilustra a articulação entre estes diferentes.

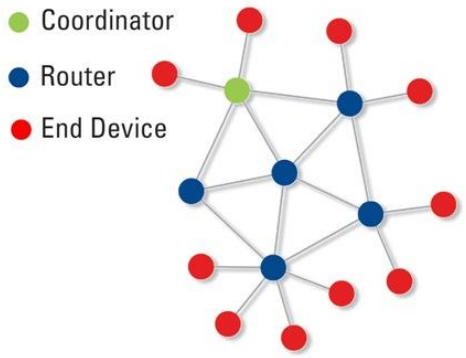



**Figura 2.3** – Estrutura genérica de uma rede *ZigBee* (fonte: http://m.eet.com/media/1072665/FIGURE_01_TI.jpg).

Todos os dispositivos *ZigBee* devem ser capazes de se juntar a uma rede, bem como abandoná-la, mas apenas o *ZigBee coordinator*, ou um *router* têm capacidade de aceitar um pedido de junção à rede. O *ZigBee coordinator* é o único capaz de iniciar a formação de uma nova rede. Os *ZigBee coordinators* e *routers* asseguram as funcionalidades adicionais de atribuição de endereços lógicos de rede e de manutenção de uma lista dos dispositivos vizinhos.

### 2.4.3. Topologias de Rede

O *ZigBee* suporta as topologias de rede apresentadas na **Figura 2.4**: estrela, ponto a ponto (*peer-to-peer* – da qual a *mesh* é um caso particular) e *cluster tree*.

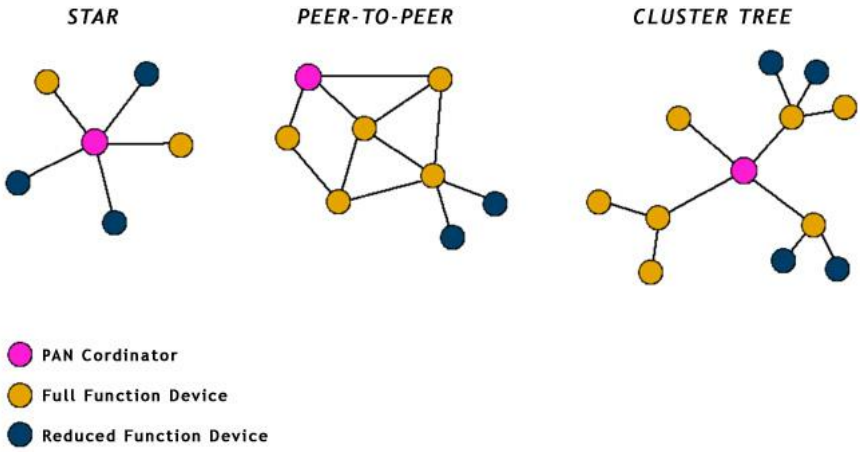



**Figura 2.4** – Topologias de rede *ZigBee* (fonte: http://wireless.arcada.fi/MOBWI/images/drawings/zigbee1.jpg).

Na topologia em estrela, a comunicação é estabelecida entre os dispositivos e um único controlador central, o *ZigBee coordinator* (ou coordenador da PAN). Depois de um FFD ser activado pela primeira vez, pode estabelecer a sua própria rede e tornar-se o coordenador da PAN. Cada rede criada escolhe um identificador da PAN (tipicamente este definido aquando da configuração dos módulos *ZigBee*), que não esteja de momento a ser usado por qualquer outra rede dentro da esfera de influência. Isto permite que cada rede em estrela possa operar independentemente.

Tipicamente, as aplicações que beneficiam desta configuração incluem domótica, ligação de PC a periféricos e brinquedos.

Na topologia *peer-to-peer*, existe também um coordenador da PAN. Ao contrário da topologia em estrela, qualquer dispositivo pode comunicar com outro, desde que esteja ao seu alcance. Uma rede deste tipo pode ser *ad hoc* (i.e. criada automaticamente para um determinado efeito), *self-organizing* (i.e. com capacidade de se organizar autonomamente para esse efeito) e *self-healing* (i.e. capaz de se reorganizar para contornar a eventual falha de um nó da rede). Permite também *multi hopping* no encaminhamento de mensagens de um dispositivo para qualquer outro na rede e um reforço da fiabilidade através de encaminhamento múltiplo (*multipath*).

Aplicações como controlo e monitorização industrial, redes de sensores ou monitorização de inventário são exemplos que beneficiam desta topologia.

A topologia *cluster tree* é um caso especial que consiste num número de redes em estrela cujos nós centrais comunicam directamente com o (único) coordenador da rede (*ZigBee coordinator*). Qualquer dos FFD pode operar como um coordenador local (*ZigBee router*) e fornecer serviços de sincronização a outros dispositivos e coordenadores.

O coordenador da PAN forma o primeiro *cluster* definindo-se como *cluster head* (CLH) com o identificador de *cluster* zero, escolhendo um identificador da PAN disponível, e enviando em *broadcast* (i.e. para todos os elementos da rede) tramas *beacon*[1] para os dispositivos vizinhos. Um dispositivo candidato que receba o *beacon* pode pedir ao CLH para se juntar à rede. Se o coordenador da PAN permitir a junção, adiciona este novo nó como um dispositivo 'filho' na sua lista de vizinhos. O novo nó adiciona o CLH como 'pai' na sua lista de vizinhos e começa a transmitir *beacons* periódicos, de maneira que outros dispositivos candidatos possam então ligar-se à rede por aquele dispositivo. Assim que os requisitos de rede ou de aplicação estejam cumpridos, o coordenador da PAN pode ordenar que um dispositivo se torne o CLH de um novo *cluster* adjacente ao primeiro.

A vantagem desta estrutura (*Cluster Tree*) consiste no aumento da área de cobertura; porém, acarreta um aumento da latência das mensagens.

## 2.5. Sistema de comunicações implementado

### 2.5.1. Razões da escolha do protocolo *ZigBee*

Neste projecto, o veículo móvel faz a aquisição sensorial e executa tarefas (comportamentos) de baixo nível (como o seguimento de trajectórias), enquanto o nó central (computador ou outro elemento com elevada capacidade de processamento) executa o algoritmo de navegação. Este algoritmo utiliza os dados sensoriais adquiridos para estimar (e representar) a posição do veículo em cada momento, num mapa previamente conhecido. É assim necessário enviar dados sensoriais do veículo para o computador, e instruções no sentido inverso.

Num contexto como o descrito, a comunicação entre o veículo (ou veículos) e o computador não precisa de ser muito complexa nem requer taxa de transmissão de dados muito elevada. Impõe-se antes que seja simples, versátil, de baixo custo e principalmente de reduzido consumo energético, pois o veículo móvel deve ter a sua própria alimentação independente (baterias) que deve durar o máximo possível.

Por isso, entre as várias alternativas analisadas (vide **Tabela 2.1** e **Figura 2.1**), o *ZigBee* apresentou-se como a mais interessante.

Acresce que se trata de uma solução utilizada em muitas outras aplicações, pelo que existem vasta informação técnica e ferramentas úteis disponíveis, entre elas uma biblioteca *open source* para *Arduino*, o microcontrolador utilizado no veículo. Isto facilita muito a utilização desta tecnologia, em particular com os módulos *ZigBee* utilizados, descritos na próxima secção, que constituíram uma escolha natural por já estarem disponíveis no laboratório em que decorreu o trabalho.

[1] Trama enviada (periodicamente) pelo *ZigBee coordinator* ou *routers* em *braodcast* informando os elementos da rede do estado da mesma.

Em resumo, o *ZigBee* vai precisamente de encontro aos requisitos deste projecto, tendo-se confirmado, como se demonstrará, a sua grande aplicabilidade em robótica móvel cooperativa.

## 2.5.2. Módulos utilizados: *XBee Series 2*

### 2.5.2.1. Características

Neste trabalho foram usados módulos rádio *XBee Series 2* como o apresentado na **Figura 2.5**, fabricados pela *Digi International*.

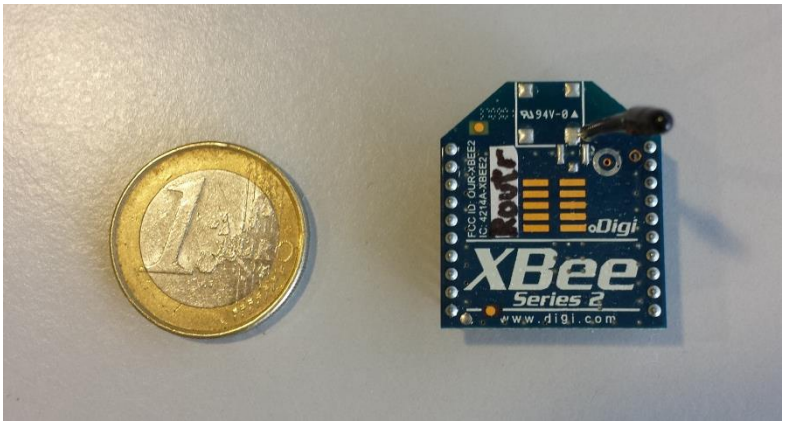



**Figura 2.5** – Módulo *XBee Series 2* utilizado.

Dotados *firmware ZigBee*, estes módulos operam na gama de frequência de 2.4GHz ISM e oferecem as características visadas de alto desempenho, baixo custo, baixo consumo e facilidade de utilização **[15]**:

- Alcance *Indoor*/urbano até 40 m
- Alcance *outdoor* em linha de visão até 120 m
- Potência de transmissão de 2 mW (3 dBm)
- Sensibilidade do receptor de -96 dBm
- *Retries* e *acknowledges* (isto é, tentativas de retransmissão e confirmação de recepção de informação)
- DSSS (*Direct Sequence Spread Spectrum*)
- Mais de 65000 endereços de rede disponíveis em cada canal de sequência directa
- Suporte a topologias *point-to-point*, *point-to-multipoint* e *peer-to-peer*
- Rede *mesh self-routing*, *self-healing* e *fault-tolerant*
- Pico de corrente no envio de 40 mA (@3.3V)
- Corrente na recepção de 40 mA (@3.3V)
- Corrente enquanto desligado <1 µA
- Comunicações RF *"out-of-box"* sem necessidade de configuração
- Modos de comando AT e API para configuração dos parâmetros dos módulos
- Tamanho reduzido
- Vasto conjunto de comandos

- Fácil configuração com o *software* X-CTU fornecido gratuitamente pelo fabricante
- Suporte técnico gratuito e ilimitado

Usaram-se *shields* para facilitar as conexões. O utilizado no módulo *XBee* ligado ao *Arduino* permite encaixá-lo numa *breadboard* e utilizar, dos seus 20 pinos, apenas os 4 necessários. Permite ainda obter a alimentação do módulo (3.3V) a partir de uma fonte de 5V. No caso do módulo ligado ao computador, usa-se um *shield* para conexão via USB.

Estes módulos são muito simples de usar; a transmissão (envio e recepção) de dados é realizada por uma porta série, sob a forma de sequências de *bytes* organizados segundo a trama definida pelo fabricante. Bastam, por isso, além dos de alimentação (VCC e GND), 2 pinos: DIN e DOUT. A interface usada é a UART.

A retransmissão de pacotes pela rede *ZigBee*, se necessária, é feita pelos próprios nós da rede. A única responsabilidade do utilizador/programador é garantir que a configuração dos vários nós da rede é feita correctamente, todos pertencem à mesma PAN e estão ao alcance uns dos outros, isto é, que existe pelo menos um caminho entre os nós que possibilita a comunicação entre todos os elementos da rede.

#### 2.5.2.2. Transmissões de Dados e Descrição da Trama

As comunicações entre os módulos podem servir para trocas de dados propriamente ditos ou para criação, gestão e manutenção da rede. Existem, por isso, vários tipos de tramas. As tramas têm tamanho variável mas campos fixos (de acordo com o tipo de trama).

Todos os módulos *XBee* têm um endereço único de 64bit atribuído durante a fabricação, designado ***extended address***. Quando se ligam a uma rede *ZigBee*, é-lhes atribuído, pelo router ou coordenador da rede, um endereço adicional de 16bit, chamado ***network address***. Este endereço é gerado aleatoriamente. O endereço de 16bit 0x0000 é reservado ao próprio coordenador da PAN.

O endereço de rede (16bit) pode mudar sob certas circunstâncias:

- Quando é detectado um conflito de endereços (dois dispositivos com o mesmo endereço de rede);
- Quando um dispositivo abandona a rede e mais tarde volta a ligar-se a ela.

Todas as transmissões *ZigBee* são feitas enviando na trama os endereços de 16bit da fonte e do destinatário. As tabelas de encaminhamento dos nós da rede usam também os endereços de 16bit para determinar o caminho que os pacotes de dados devem tomar pela rede. Porém, dado que estes endereços não são estáticos, esta não é uma maneira fiável para endereçar um dispositivo. Para contornar este problema e garantir que o pacote é entregue ao destinatário correcto, o endereço único (64bit) pode também ser especificado na trama de dados.

Os pacotes de dados *ZigBee* podem ser enviados em *broadcast* ou em *unicast*.

As transmissões *broadcast* (vide **Figura 2.6**) são propagadas para chegar a todos os nós da rede. Neste processo, o coordenador e todos os *routers* que recebam o pacote de

dados reenviam-no três vezes, excepto no caso de nós filhos (*end devices*), para os quais o enviam uma única vez.

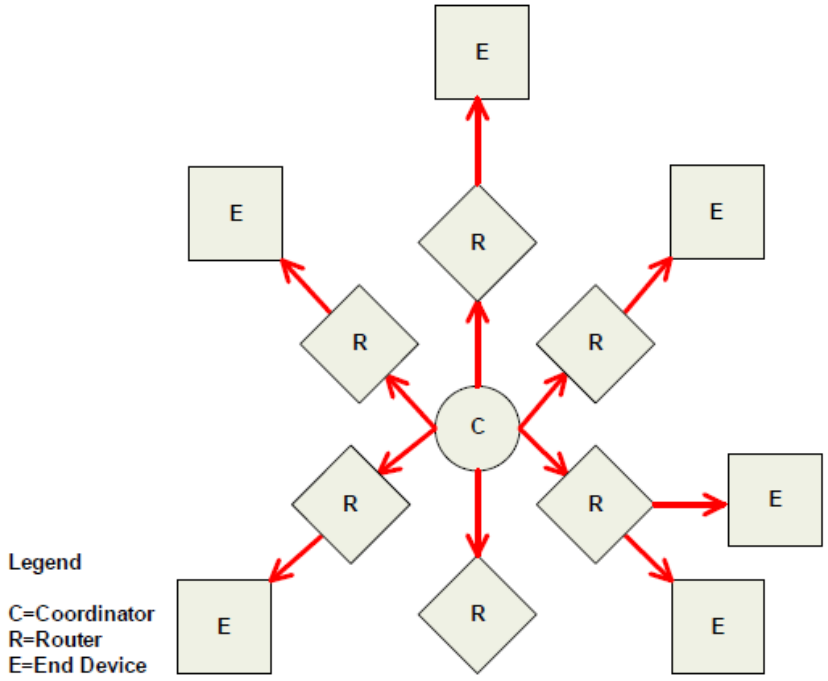



**Figura 2.6** – Esquema ilustrativo de uma transmissão *broadcast* **[15]**.

Quando um nó faz *broadcast* de um pacote de dados, regista o evento numa tabela de transmissões *broadcast*. Este historial das transmissões mais recentes permite garantir que elas não são repetidas indefinidamente. Cada registo permanece na tabela por 8 segundos, e a tabela permite 8 registos.

As transmissões *unicast* são enviadas para um nó específico. Este pode ser um vizinho imediato da fonte, ou pode estar a vários "saltos" de distância. No 2º cenário, é preciso estabelecer um caminho entre fonte e destino.

Quando é enviado um pacote de dados *unicast*, a camada de rede *ZigBee* usa o endereço de rede do dispositivo destino para reencaminhar o pacote em cada "salto". Caso este não seja conhecido, a *stack ZigBee* aplica um método de descoberta para obter o endereço de 16bit do dispositivo destino antes de reencaminhar o pacote de dados. Neste método, o dispositivo que inicia o processo envia uma transmissão *broadcast* de descoberta de endereço. Esta inclui o endereço de 64bit do dispositivo remoto cujo endereço de 16bit está a ser solicitado. Cada nó que recebe esta transmissão compara o endereço de 64bit contido no pacote de dados com o seu próprio endereço de 64bit; se forem coincidentes, o envia um pacote de resposta ao que iniciou a procura. Quando este recebe a resposta, envia finalmente os dados. Quando se pretende enviar um pacote de dados para um destino cujo endereço de rede é desconhecido, o campo respectivo da trama deve ser preenchido com 0xFFFE.

Cada dispositivo *ZigBee* mantém actualizada uma tabela de endereços que associa um endereço de rede (16bit) a cada endereço único (64bit), como se pode ver na **Tabela 2.2.** Quando uma transmissão é endereçada a um dispositivo pelo seu endereço de 64bit, a *stack*

*ZigBee* procura na tabela o endereço de 16bit correspondente. Se for encontrado, é colocado no campo respectivo na trama de dados; caso contrário, esse campo é preenchido com o valor 0xFFFE e executado o processo de descoberta de endereço.

**Tabela 2.2** – Exemplo de tabela de endereços de um nó da rede **[15]**.

Sample Address Table

| 64-bit Address | 16-bit Address |
|---|---|
| 0013 A200 4000 0001 | 0x4414 |
| 0013 A200 400A 3568 | 0x1234 |
| 0013 A200 4004 1122 | 0xC200 |
| 0013 A200 4002 1123 | 0xFFFE (unknown) |

Um módulo *XBee* pode armazenar até 10 entradas na sua tabela de endereços. Em aplicações em que sejam enviados pacotes para mais de 10 destinatários, a implementação de uma tabela de endereços capaz de armazenar mais de 10 entradas fica a cargo do programador. Existem diversos tipos de tramas em que o dispositivo fonte envia também o seu endereço de 16bit; uma aplicação eficiente deve utilizar essa informação para manter actualizada a tabela de endereços. Em caso de falha na entrega de um pacote de dados, deve preencher o endereço de 16bit do destinatário respectivo com 0xFFFE.

Cada transmissão *unicast* suporta até 84 bytes de dados úteis (*payload*[2]). No entanto, o *firmware XBee* ZB permite fragmentação, isto é, possibilita o envio de pacotes de dados de tamanho superior, divididos por múltiplas transmissões RF e reagrupados no receptor antes de serem enviados para a aplicação pela UART. A **Figura 2.7** ilustra este processo.

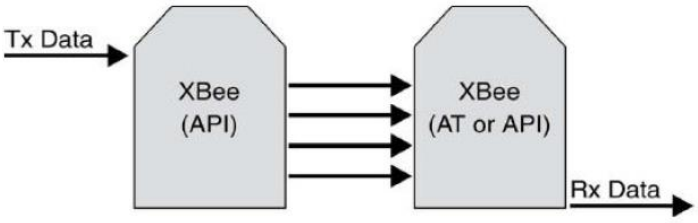



**Figura 2.7** – Fragmentação **[15]**.

Recorrendo a fragmentação, trama de envio de dados pode incluir até 255 bytes de *payload*. Se um ou mais fragmentos da mensagem original forem perdidos, o receptor descarta todos os fragmentos, informa o evento ao emissor, e este indica um erro de falha de transmissão na trama de estado de envio.

Os módulos *XBee* podem operar em dois modos: AT – "*Transparent Operation*" – e API – "*Application Programming Interface (Operation)*". O modo AT é mais simples de usar; funciona basicamente como uma porta série, isto é, os dados pretendidos são enviados

[2] Campo da trama ZigBee onde são colocados os bytes de dados propriamente ditos.

directamente; no entanto, este modo de funcionamento apenas suporta a comunicação de um dispositivo para outro. Já o modo API permite a comunicação entre diversos dispositivos dentro da mesma rede. No entanto, os dados devem ser organizados numa trama de estrutura bem definida. O envio de dados de e para os módulos é feito usando uma trama de dados UART.

Neste projecto foi utilizado o modo API, mas inicialmente foram feitos alguns testes no modo AT para familiarização com o equipamento antes de prosseguir para o modo API, que é mais complexo.

Dentro do modo API, existem dois modos: "*API Operation*" e "*API Operation with escaped characters*". A diferença entre eles consiste no seguinte: a trama tem campos de valor fixo, tal como o *byte* que indica o início de uma trama (0x7E); para evitar que *bytes* de dados com esses valores possam ser erradamente interpretados, o segundo o modo de operação prevê um mecanismo de os "mascarar", enquanto o primeiro envia a trama com os valores originais sem qualquer tipo de alteração.

Neste trabalho, foi utilizado o modo de operação com *escaped characters*. Para fazer o "*escape*" de um *byte* de dados, este é precedido por 0x7D e é feita a operação lógica XOR entre o *byte* em causa e o valor 0x20. São quatro os valores que necessitam de ser "mascarados":

- 0x7E – "*Frame Delimiter*"
- 0x7D – "*Escape*"
- 0x11 – XON
- 0x13 – XOFF

O formato genérico das tramas usadas neste modo é apresentado na **Figura 2.8**.

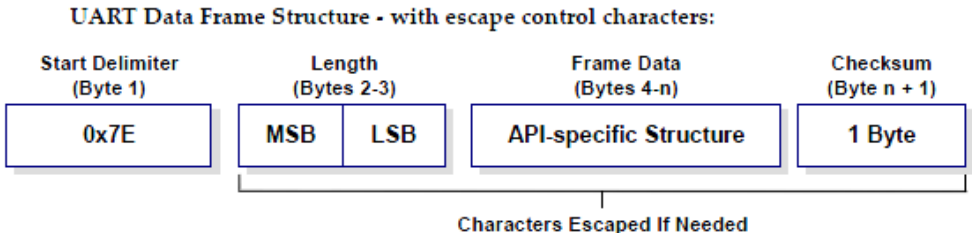



**Figura 2.8** – Trama de comunicações dos rádios *XBee* **[15]**.

Todas as tramas, independentemente do tipo, têm 3 campos comuns:

- O *byte* delimitador do início de uma trama (0x7E);
- Dois *bytes* indicadores do tamanho da trama (entendido como o número de *bytes* indicadores do *Frame Data*, isto é, entre os campos *tamanho* e *checksum*, exclusive).
- Um *byte* para *checksum*, destinado a detectar eventuais erros na transmissão; a sua fórmula é:

$$checksum = 0xFF - \sum(todos\ os\ bytes\ após\ 'Length') \qquad 2.1$$

O campo *Frame Data* é específico de cada tipo de trama mas, o seu primeiro *byte* é sempre o identificador do tipo de trama. Os campos da trama são preenchidos segundo o método *big endian*[3].

Os módulos *XBee* suportam uma extensa lista de tipos de trama, apresentados na Tabela 2.3.

**Tabela 2.3** – Tramas suportadas pelos módulos *XBee*.

**API Frame Names and Values**

| API Frame Names | API ID |
|---|---|
| AT Command | 0x08 |
| AT Command - Queue Parameter Value | 0x09 |
| ZigBee Transmit Request | 0x10 |
| Explicit Addressing ZigBee Command Frame | 0x11 |
| Remote Command Request | 0x17 |
| Create Source Route | 0x21 |
| AT Command Response | 0x88 |
| Modem Status | 0x8A |
| ZigBee Transmit Status | 0x8B |
| ZigBee Receive Packet (AO=0) | 0x90 |
| ZigBee Explicit Rx Indicator (AO=1) | 0x91 |
| ZigBee IO Data Sample Rx Indicator | 0x92 |
| XBee Sensor Read Indicator (AO=0) | 0x94 |
| Node Identification Indicator (AO=0) | 0x95 |
| Remote Command Response | 0x97 |
| Over-the-Air Firmware Update Status | 0xA0 |
| Route Record Indicator | 0xA1 |
| Many-to-One Route Request Indicator | 0xA3 |

Aqui, centrar-nos-emos nas mais relevantes neste projecto: *ZigBee Transmit Request*, *ZigBee Transmit Status* e *ZigBee Receive Packet*. Tal como os nomes indicam, estas tramas servem, respectivamente, para enviar dados, obter o estado de envio do pacote, e receber um pacote de dados. Os seus papéis estão esquematizados na **Figura 2.9**.

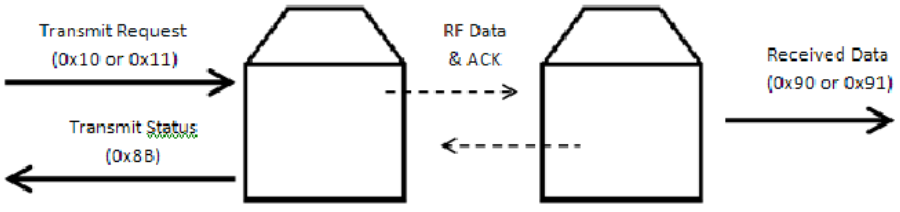



**Figura 2.9** –Tipos de tramas envolvidas na transmissão de um pacote de dados **[15]**.

- *ZigBee Transmit Request*

Este tipo de trama tem o identificador 0x10 e serve para que o módulo envie dados como um pacote RF para o destino especificado. No caso de se pretender enviar o pacote em *broadcast*, o endereço destino de 64bit deve ser definido como 0x000000000000FFFF.

[3] Método de organizar valores binários, em que os bytes mais significativos são preenchidos primeiro (endereços de memória mais baixos) e os menos significativos no final.

Para se dirigir um pacote para o coordenador da PAN, pode-se preencher endereço de 64bit com 0x0000000000000000 e o de 16bit com 0xFFFE, ou preencher o endereço de 64bit com o endereço único do dispositivo e o de 16bit com 0x0000. Para todas as outras transmissões, é vantajoso indicar o valor correcto do endereço de 16bit, pois isso melhora o desempenho quando se transmite para múltiplos destinos. Caso não seja conhecido, preenche-se com 0xFFFE; se o envio dos dados for bem sucedido, o endereço correcto será indicado na trama *ZigBee Transmit Status*. A trama completa é apresentada na Tabela 2.4, já preenchida com alguns valores a título de exemplo.

**Tabela 2.4** – Estrutura de uma trama *ZigBee Transmit Request* **[15]**.

| | Frame Fields | | Offset | Example | Description |
|---|---|---|---|---|---|
| API Packet | Start Delimiter | | 0 | 0x7E | |
| | Length | | MSB 1 | 0x00 | Number of bytes between the length and the checksum |
| | | | LSB 2 | 0x16 | |
| | Frame-specific Data | Frame Type | 3 | 0x10 | |
| | | Frame ID | 4 | 0x01 | Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent. |
| | | 64-bit Destination Address | MSB 5 | 0x00 | Set to the 64-bit address of the destination device. The following addresses are also supported:<br>0x0000000000000000 - Reserved 64-bit address for the coordinator<br>0x000000000000FFFF - Broadcast address |
| | | | 6 | 0x13 | |
| | | | 7 | 0xA2 | |
| | | | 8 | 0x00 | |
| | | | 9 | 0x40 | |
| | | | 10 | 0x0A | |
| | | | 11 | 0x01 | |
| | | | LSB 12 | 0x27 | |
| | | 16-bit Destination Network Address | MSB 13 | 0xFF | Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast. |
| | | | LSB 14 | 0xFE | |
| | | Broadcast Radius | 15 | 0x00 | Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value. |
| | | Options | 16 | 0x00 | Bitfield of supported transmission options. Supported values include the following:<br><br>0x01 - Disable retries and route repair<br>0x20 - Enable APS encryption (if EE=1)<br>0x40 - Use the extended transmission timeout<br><br>Enabling APS encryption presumes the source and destination have been authenticated. I also decreases the maximum number of RF payload bytes by 4 (below the value reported by NP).<br><br>The extended transmission timeout is needed when addressing sleeping end devices.It also increases the retry interval between retries to compensate for end device polling.See Chapter 4, Transmission Timeouts, Extended Timeout for a description.<br><br>Unused bits must be set to 0. |
| | | RF Data | 17 | 0x54 | Data that is sent to the destination device |
| | | | 18 | 0x78 | |
| | | | 19 | 0x44 | |
| | | | 20 | 0x61 | |
| | | | 21 | 0x74 | |
| | | | 22 | 0x61 | |
| | | | 23 | 0x30 | |
| | | | 24 | 0x41 | |
| | Checksum | | 25 | 0x13 | 0xFF - the 8 bit sum of bytes from offset 3 to this byte. |

- *ZigBee Transmit Status*

Esta trama indica se o pacote foi enviado com sucesso ou se surgiu alguma falha. O seu identificador é 0x8B. A estrutura desta trama corresponde sempre a 11 campos de 1 *byte*, o que resulta num tamanho fixo de 11 *bytes*. O esquema detalhado da trama e de cada campo encontra-se na Tabela 2.5.

**Tabela 2.5** – Estrutura de uma trama *ZigBee Transmit Status* **[15]**.

| | Frame Fields | | Offset | Example | Description |
|---|---|---|---|---|---|
| API packet | Start Delimiter | | 0 | 0x7E | |
| | Length | | MSB 1 | 0x00 | Number of bytes between the length and the checksum |
| | | | LSB 2 | 0x07 | |
| | Frame-specific Data | Frame Type | 3 | 0x8B | |
| | | Frame ID | 4 | 0x01 | Identifies the UART data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given. |
| | | 16-bit address of destination | 5 | 0x7D | 16-bit Network Address the packet was delivered to (if success). If not success, this address matches the Destination Network Address that was provided in the Transmit Request Frame. |
| | | | 6 | 0x84 | |
| | | Transmit Retry Count | 7 | 0x00 | The number of application transmission retries that took place. |
| | | Delivery Status | 8 | 0x00 | 0x00 = Success<br>0x01 = MAC ACK Failure<br>0x02 = CCA Failure<br>0x15 = Invalid destination endpoint<br>0x21 = Network ACK Failure<br>0x22 = Not Joined to Network<br>0x23 = Self-addressed<br>0x24 = Address Not Found<br>0x25 = Route Not Found<br>0x26 = Broadcast source failed to hear a neighbor relay the message<br>0x2B = Invalid binding table index<br>0x2C = Resource error lack of free buffers, timers, etc.<br>0x2D = Attempted broadcast with APS transmission<br>0x2E = Attempted unicast with APS transmission, but EE=0<br>0x32 = Resource error lack of free buffers, timers, etc.<br>0x74 = Data payload too large |
| | | Discovery Status | 9 | 0x01 | 0x00 = No Discovery Overhead<br>0x01 = Address Discovery<br>0x02 = Route Discovery<br>0x03 = Address and Route<br>0x40 = Extended Timeout Discovery |
| | Checksum | | 10 | 0x71 | 0xFF - the 8 bit sum of bytes from offset 3 to this byte. |

- *ZigBee Receive Packet*

Quando um módulo *XBee* recebe um pacote RF, este é enviado pela UART (para a aplicação) através deste tipo de trama. O seu identificador é 0x90. Tal como no caso da trama *ZigBee Transmit Request*, o seu tamanho é variável porque integra um campo de dados (*payload*) também variável. A Tabela 2.6 apresenta os campos da trama, com exemplos do seu preenchimento.

**Tabela 2.6** – Estrutura de uma trama *ZigBee Receive Packet* **[15].**

| | Frame Fields | | Offset | Example | Description |
|---|---|---|---|---|---|
| API Packet | Start Delimiter | | 0 | 0x7E | |
| | Length | | MSB 1 | 0x00 | Number of bytes between the length and the checksum |
| | | | LSB 2 | 0x11 | |
| | Frame-specific Data | Frame Type | 3 | 0x90 | |
| | | 64-bit Source Address | MSB 4 | 0x00 | |
| | | | 5 | 0x13 | 64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown. |
| | | | 6 | 0xA2 | |
| | | | 7 | 0x00 | |
| | | | 8 | 0x40 | |
| | | | 9 | 0x52 | |
| | | | 10 | 0x2B | |
| | | | LSB 11 | 0xAA | |
| | | 16-bit Source Network Address | MSB 12 | 0x7D | 16-bit address of sender |
| | | | LSB 13 | 0x84 | |
| | | Receive Options | 14 | 0x01 | 0x01 - Packet Acknowledged<br>0x02 - Packet was a broadcast packet<br>0x20 - Packet encrypted with APS encryption<br>0x40 - Packet was sent from an end device (if known)<br>Note: Option values can be combined. For example, a 0x40 and a 0x01 will show as a 0x41. Other possible values 0x21, 0x22, 0x41, 0x42, 0x60, 0x61, 0x62. |
| | | Received Data | 15 | 0x52 | Received RF data |
| | | | 16 | 0x78 | |
| | | | 17 | 0x44 | |
| | | | 18 | 0x61 | |
| | | | 19 | 0x74 | |
| | | | 20 | 0x61 | |
| | Checksum | | 21 | 0x0D | 0xFF - the 8 bit sum of bytes from offset 3 to this byte. |

### 2.5.2.3. Configuração dos Módulos *XBee*

A configuração dos módulos rádio *XBee* requer o envio de uma sequência adequada de caracteres, o que pode ser feito a partir do computador de duas maneiras:

- Ligando o módulo via porta série e usando um terminal;
- Ligando o módulo por USB através do *shield* adequado; neste caso, pode usar-se um terminal ou utilizar o programa X-CTU, disponibilizado pelo fabricante. Foi adoptada esta última alternativa, por ser mais simples e intuitiva.

Com o programa X-CTU, especificar o papel do dispositivo na rede, o seu modo de funcionamento (AT ou API) e editar todos os parâmetros relevantes (entre os que é possível editar, pois alguns são apenas de leitura e não de escrita), tais como o identificador da rede (PAN ID) ou modo de operação dentro do API. É também possível emular um terminal e receber/enviar mensagens em código ASCII ou hexadecimal.

A Figura 2.10 mostra um exemplo de configuração de um módulo *XBee*. Neste caso, trata-se do coordenador, que está conectado ao PC. Da figura destacam-se a função do módulo ("***Function Set***") e a **PAN ID**.

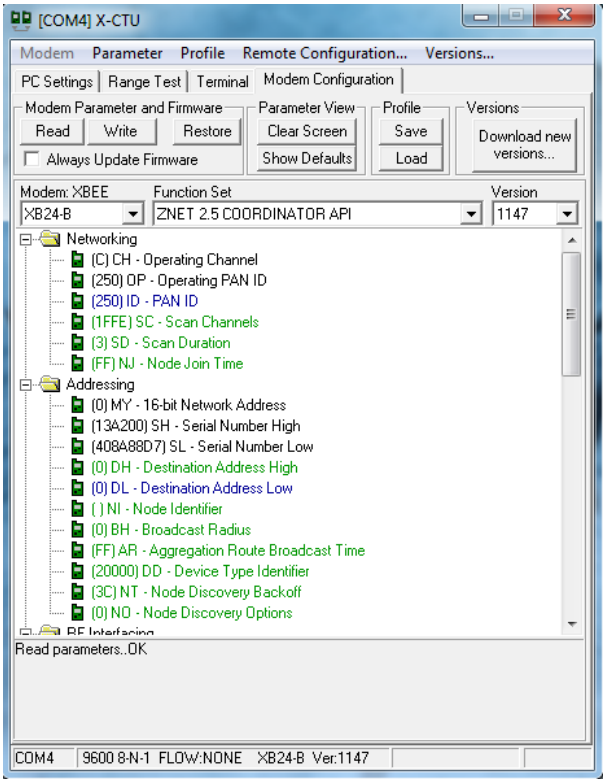



**Figura 2.10** - Exemplo de configuração de um módulo *XBee*. Destacam-se: a função do dispositivo, o PAN ID e o endereço de 64bit do módulo (campos SH e SL).

## 2.6. Testes de comunicação

Os testes apresentados aqui destinaram-se a configurar e avaliar o desempenho das comunicações *ZigBee* entre dispositivos dos tipos que se perspectivava utilizar no controlo e navegação do veículo robótico a construir. Assim, envolveram:

- Computador portátil;
- Microcontroladores *Arduino Uno Rev3* e *Arduino Duemilanove*;
- 4 Módulos *XBee Series 2* da *Digi International*;
- *Shields* para montagem dos módulos em placa branca e ligação ao computador via USB;
- *Software* X-CTU para configuração dos módulos, e visualização (no terminal) das mensagens trocadas nas comunicações.

Foi ainda utilizada uma biblioteca *open source* para utilização dos módulos *XBee* em *Arduino*. Esta biblioteca trata da construção automática das tramas, bem como o seu envio e recepção, de acordo com a estrutura já apresentada (**Figura 2.8**), bastando definir, ao invocar as funções no código do *Arduino*, os endereços de destino e os dados a enviar. No PC, foram construídas funções em *Matlab* para envio e recepção de tramas, sem recurso a qualquer biblioteca; nestas funções, as tramas são construídas/enviadas e recebidas/analisadas *byte* a *byte*.

Na **Tabela 2.7**, encontra-se um resumo dos cenários de teste e seus resultados qualitativos. A ordem na tabela respeita simplesmente a ordem cronológica de execução;

não reflecte critérios de importância ou relevância para a aplicação em vista (controlo e navegação de um veículo robótico).

**Tabela 2.7** – Resumo qualitativo dos resultados dos testes à comunicação entre módulos *XBee*.

| Teste | | | Comunicação Assegurada | Perda de Dados |
|---|---|---|---|---|
| 1. *Arduino* -> PC | | | ✓ | Não |
| 2. PC -> *Arduino* | | | ✓ | Não |
| 3. *Arduino* -> *Arduino* | | | ✓ | Não |
| 4. PC -> PC | | | ✓ | Não |
| 5. PC -> Dois *Arduinos* | | | ✓ | Não |
| 6. PC -> Dois *Arduinos* (com Retransmissão) | | | ✓ | Não |
| 7. Dois *Arduinos* (envio simultâneo a taxas iguais) -> PC | 7.1. | 1Hz | ✓ | Não |
| | 7.2. | 2Hz | ✓ | Não |
| | 7.3. | 4Hz | ✓ | Não |
| | 7.4. | 10Hz | ✓ | Não |
| | 7.5. | 20Hz | ✓ | Não |
| | 7.6. | 100Hz | ✓ | Não |
| | 7.7. | Taxa max | ✓ | Sim |

Segue-se uma apresentação detalhada de cada teste. Em todos, o endereçamento dos dispositivos recorreu ao seu endereço de 64bit; nunca foram usados endereços de 16bit.

**Teste 1. *Arduino* -> PC**

Este primeiro teste serviu para testar a comunicação entre o *Arduino Uno* e a estação central – o computador portátil. Consistiu no envio periódico, via *XBee*, de valores analógicos lidos do pino A5 do *Arduino* e visualização dos pacotes recebidos no PC com ajuda do programa X-CTU (no terminal). Note-se que apenas se pretendeu verificar a comunicação entre os dois módulos, pelo que o valor analógico lido era irrelevante: não se ligou qualquer elemento ao pino A5.

No *Arduino* executou-se o código de exemplo **Series2_Tx** (da biblioteca *XBee* para *Arduino*) que lê periodicamente um valor analógico do pino A5 do *Arduino*, encapsula-o em 2 *bytes* (pois a ADC é de 10 *bits*) numa trama de dados e envia-o numa trama de dados para o módulo *XBee* conectado ao *Arduino*. Este envia o pacote RF para o módulo *XBee* definido como destino. No PC, está aberto o terminal do programa X-CTU na porta COM correspondente ao seu módulo *XBee*.

Os módulos *XBee* (um ligado ao PC e o outro ao *Arduino*) foram configurados recorrendo ao *software* X-CTU. Apenas se editaram alguns parâmetros, deixando os restantes com os valores *default*. Refira-se que *Baudrate* foi mantido no valor *default* de 9600. Os parâmetros alterados em cada módulo foram os seguintes:

- Módulo *XBee* no *Arduino* (endereço único 0x0013A200408A88D7):
  - Modo de Operação: ZNET 2.5 COORDINATOR API
  - PAN ID = 250
  - AP = 2 (“*escaped characters enabled*”)
  - A0 = 0 (“*options*”)
- Módulo *XBee* no PC (endereço único 0x0013A200405C4EC5):

- Modo de Operação: ZNET 2.5 ROUTER/END DEVICE API
- PAN ID = 250
- AP = 2
- A0 = 0

O PAN ID poderia ser qualquer dentro da gama possível, desde que igual em ambos os módulos.

Apresentam-se três exemplos de tramas recebidas no terminal (em hexadecimal) da porta COM onde está ligado o *XBee* ao PC.

- 7E000E90007D33A200408A88D700000101543B
- 7E000E90007D33A200408A88D700000101632C
- 7E000E90007D33A200408A88D700000101622D

Podemos desde já fazer duas observações: todas começam com o delimitador 0x7E, como seria de esperar; existem *bytes* 0x7D, o que significa que os *bytes* seguintes foram "mascarados", isto é, o valor original sofreu a operação de XOR com 0x20. Para melhor interpretar as tramas, a **Tabela 2.8** apresenta a sua divisão pelos campos do tipo *ZigBee Receive Packet*, dado que são pacotes recebidos no PC (provenientes do *Arduino*).

**Tabela 2.8** – Tramas recebidas no PC, provenientes do *Arduino*.

| Start | Length | Frame type | 64bit Address (sender) | 16bit Address (sender) | Receive Options | Received RF Data | Checksum |
|---|---|---|---|---|---|---|---|
| 7E | 000E | 90 | 0013A200408A88D7 | 0000 | 01 | 0154 | 3B |
| 7E | 000E | 90 | 0013A200408A88D7 | 0000 | 01 | 0163 | 2C |
| 7E | 000E | 90 | 0013A200408A88D7 | 0000 | 01 | 0162 | 2D |

Como seria de esperar, as tramas são idênticas excepto no campo de dados, que contém o valor analógico lido do pino A5 do *Arduino*, e no campo *checksum*, por depender do primeiro.

Eis a análise detalhada dos valores de cada campo:

- *Length*: é o número de *bytes* do pacote entre *length* e *checksum*; o seu valor decimal é **14**, pois há **1** *byte* para o *Frame Type*, **8** *bytes* no campo *64bit Address*, **2** *bytes* no campo *16bit Address*, **1** *byte* para as opções e **2** bytes de dados recebidos.
- *Frame Type*: identifica o tipo de trama; o seu valor hexadecimal é 0x90, ou seja, confirma-se que são tramas do tipo *ZigBee Receive Packet*.
- *64bit Address (sender)*: é o endereço de 64bit do dispositivo que enviou o pacote; de facto, contém o endereço do módulo *XBee* ligado ao *Arduino*.
- *16bit Address (sender)*: é o endereço de rede do dispositivo que enviou o pacote; como o módulo *XBee* ligado ao *Arduino* foi configurado como coordenador da PAN, é-lhe reservado o endereço 0x0000.
- *Receive Options*: define as opções da recepção, e pode tomar os valores apresentados na Tabela 2.6. Neste caso, o valor 0x01 significa "*Packet Acknowledged*", isto é, foi enviada uma confirmação de recepção de pacote ao dispositivo que o enviou.

- *Received RF Data*: contém os dados propriamente ditos. Estes valores encontram-se na gama esperada (0 a 1023) atendendo a que os conversores analógico-digitais (ADC) do *Arduino* são de 10 *bits*.
- *Checksum*: é construído no emissor do pacote. Se o receptor detectasse inconsistência, descartaria esse pacote; por isso, a recepção é prova de que o campo *checksum* foi correctamente construído. Com efeito, aplicando a expressão **2.1**, confirmam-se os valores observados em todos os casos.

Conclui-se que a comunicação no sentido ***Arduino* -> PC** é bem conseguida, sendo a sua implementação simplificada pelo uso da biblioteca *XBee* para *Arduino*.

**Teste 2. PC -> *Arduino***

Neste teste, pretendeu-se testar a comunicação no sentido inverso ao do teste anterior. Do lado do PC enviaram-se pacotes de dados; no *Arduino*, como não se consegue consultar directamente os dados recebidos, recorreu-se ao *led* incorporado para identificar a recepção bem-sucedida de um pacote *XBee*, fazendo-o piscar um certo número de vezes sempre que tal acontecia.

Construíram-se manualmente algumas tramas (não dando importância ao campo de dados, pois o *led* no *Arduino* iria piscar com a recepção do pacote, independentemente do seu conteúdo); juntaram-se todos os *bytes* num único pacote e enviou-se através do terminal do X-CTU para a porta COM ligada ao módulo *XBee*. O *Arduino* executava código adaptado do exemplo **Series2_Rx** da biblioteca que o fazia continuamente tentar ler pacotes novos. Sempre que recebia um, verificava o seu tipo e piscava o *led* se se tratasse de um *ZigBee Receive Packet*.

Os módulos *XBee* usados foram exactamente os mesmos que no teste anterior e as suas configurações mantiveram-se inalteradas, apenas com a diferença de que desta vez o módulo coordenador da PAN estava ligado ao PC e o módulo *router* ao *Arduino*.

Conforme se ilustrou **Figura 2.9**, tramas de três tipos estão envolvidas na transmissão de um pacote. No teste anterior, analisou-se a trama do tipo *ZigBee Receive Packet*. Neste teste são analisadas as outras duas, observadas no terminal do X-CTU no PC: *ZigBee Transmit Request* e *ZigBee Transmit Status*. Quando se envia a primeira trama, a segunda surge como resposta, relatando o estado do envio do pacote.

Comecemos por analisar a trama construída e enviada através do terminal. É apresentada no formato original, para mais fácil interpretação (mas antes do envio realizou-se o "*escape*" dos *bytes* necessários) e já dividida pelos campos do tipo *ZigBee Transmit Request*.

| Start | Length | Frame type | Frame ID | 64bit Address (destination) | 16bit Address (destination) | Broadcast Radius | Options | RF Data | Checksum |
|---|---|---|---|---|---|---|---|---|---|
| 7E | 0010 | 10 | 01 | 0013A200405C4EC5 | FFFE | 00 | 00 | 0161 | 2B |

Analisando individualmente cada campo da trama, temos:

- *Length*: é o número de *bytes* entre *length* e *checksum*. O campo indica 16 *bytes*, o que facilmente se confirma somando o número de *bytes* de cada campo.
- *Frame Type*: tipo de trama; neste caso o valor é 0x10 (*ZigBee Transmit Request*)
- *Frame ID*: serve para correlacionar o pacote enviado com um *acknowledge* a receber. Se for definido como 0x00, não será recebida nenhuma confirmação de recepção.
- *64bit Address (destination)*: contém o endereço de 64bit do dispositivo destino que, neste caso, é o módulo *XBee* ligado ao *Arduino*.
- *16bit Address (destination)*: como neste caso se usa o endereço de 64bit para fazer o endereçamento, é definido como 0xFFFE.
- *Broadcast Radius*: número máximo de "saltos" que um pacote pode dar para chegar ao destino; se for definido como 0x00, esse número é o máximo possível.
- *Options*: este campo permite definir algumas opções de transmissão e/ou encriptação, como é mostrado na Tabela 2.4. Neste caso, está preenchido com 0x00, o que significa que não foram definidas quaisquer opções.
- *RF Data*: contém os dados propriamente ditos. Neste teste o seu conteúdo não é relevante; transmitiram-se dois *bytes* com valor arbitrário (no caso, 0x0161).
- *Checksum*: campo para verificação da coerência da trama. Se for aplicada a expressão **2.1**, facilmente se confirma que o valor está correcto.

Após o envio desta trama através do terminal do X-CTU, recebe-se de imediato o estado do envio, numa trama *ZigBee Transmit Status* cujo conteúdo, já dividido pelos respectivos campos, é o seguinte:

| Start | Length | Frame type | Frame ID | 16bit Address (destination) | Transmit Retry Count | Delivery Status | Discovery Status | Checksum |
|---|---|---|---|---|---|---|---|---|
| 7E | 0007 | 8B | 01 | 6ACE | 00 | 00 | 00 | 3B |

Segue-se uma análise dos campos mais relevantes desta trama:

- *Frame Type*: O valor 0x8B corresponde ao tipo *ZigBee Transmit Status*.
- *Frame ID*: identifica a trama UART a ser relatada com *acknowledge* ou não.
- *16bit Address (destination)*: contém o endereço de 16bit do dispositivo destino, caso o pacote tenha sido enviado com sucesso. Se o envio tiver falhado, este campo conterá o valor especificado no envio do pacote (trama *ZigBee Transmit Request*).
- *Transmit Retry Count*: número de tentativas de retransmissão do pacote. Neste caso temos zero, o que significa que a trama foi enviada com sucesso à primeira.

- *Delivery Status*: este é o campo mais relevante, pois indica o resultado do envio do pacote. Existem diversos valores possíveis (vide Tabela 2.5); neste caso particular, tem-se 0x00, o que significa que o envio foi bem sucedido.
- *Discovery Status*: relata o estado da descoberta do dispositivo destino (ver Tabela 2.5). Neste caso, 0x00 significa "*No Discovery Overhead*", isto é, não houve descoberta do dispositivo antes de enviar o pacote, o que significa que o endereço de rede do dispositivo destino já existia na tabela de endereços do módulo *XBee* do PC no momento do envio do pacote.

Face à análise dos campos da trama, conclui-se que a comunicação do PC para o *Arduino* é bem conseguida. Isto é comprovado em ambos os nós envolvidos: do lado do *Arduino* verificou-se que quando se enviava uma trama do terminal do X-CTU no PC, o *led* piscava, sinalizando a chegada de um pacote de dados do tipo *ZigBee Receive Packet*; no lado do PC, após o envio dessa mesma trama para o *Arduino*, era recebida no terminal quase instantaneamente uma nova trama do tipo *ZigBee Transmit Status*, confirmando, como se viu, a recepção bem sucedida do pacote.

**Teste 3.** ***Arduino -> Arduino***

Este 3º teste teve como objectivo testar a comunicação entre dois *Arduinos* num sentido. Consistiu no envio periódico de um pacote de dados de um *Arduino* para o outro, que deveria sinalizar a recepção do pacote. Apesar de não ser relevante, o campo de dados do pacote enviado consiste num valor analógico lido do pino A5 do *Arduino*; não fazendo uso desse valor, o receptor sinaliza apenas a chegada do pacote de dados.

Nos *Arduinos* não é possível ver directamente as tramas enviadas/recebidas, pelo que foram utilizados os leds para interpretar o que vai acontecendo. O *Arduino Uno* (emissor), ao enviar uma trama, pisca o led de uma maneira bem definida, para identificar esse evento. O *Arduino Duemilanove* (receptor), ao receber uma trama, verifica o seu tipo e, caso se trate de uma trama *ZigBee Receive Packet*, pisca o seu led de um outro modo também definido *a priori*. Após o envio da trama, o *Arduino Uno* tenta receber um pacote de dados e verifica o seu tipo; se for *ZigBee Transmit Status*, lê o "relatório" do envio da trama e pisca o led de uma maneira em caso de sucesso na recepção por parte do *Arduino Duemilanove*, ou de outra maneira no caso contrário.

Os módulos *XBee* utilizados foram os mesmos, com a mesma configuração. No *Arduino Uno* era executado o exemplo **Series2_Tx** da biblioteca *XBee*, e tinha a si ligado o módulo *XBee* coordenador (endereço 0x0013A200408A88D7). No *Arduino Duemilanove* corria um código baseado no exemplo **Series2_Rx** e utilizava o módulo *XBee* router (endereço 0x0013A200405C4EC5).

Neste teste não se obtiveram tramas para analisar. No entanto, através dos leds de ambos os *Arduinos* foi possível confirmar que estes comunicavam: quando o *Arduino Uno* enviava um pacote, o *Arduino Duemilanove* sinalizava a recepção de uma trama do tipo *ZigBee Receive Packet*, ao mesmo tempo que o *Arduino Uno* sinalizava a recepção do *acknowledge* proveniente do *Arduino Duemilanove*.

**Teste 4.** **PC -> PC**

No 4º teste pretendeu-se testar a comunicação entre dois módulos ligados ao PC, em portas diferentes. Este cenário teve uma grande vantagem em relação aos outros: como ambos os módulos estavam ligados a portas COM, foi possível aceder a ambos através de dois terminais e ver exactamente as tramas que cada um enviava e recebia. Basicamente, este teste consistiu no envio de uma trama de um dos módulos *XBee* para o outro, usando os respectivos terminais para visualizar as tramas enviadas/recebidas.

Num dos terminais foi construída uma trama preenchendo o campo de dados com 2 bytes a título de exemplo, e enviada pela porta COM para o módulo *XBee*, que envia o pacote RF para o módulo destinatário. Seguidamente surgem quase simultaneamente duas novas tramas, uma em cada terminal: ao emissor chega a trama *ZigBee Transmit Status* com o estado do envio; ao receptor chega uma trama *ZigBee Receive Packet* com os dados enviados pelo emissor.

Os módulos *XBee* utilizados foram os mesmos dos testes anteriores, bem como as suas configurações. O módulo coordenador foi conectado à porta COM cujo terminal iria ser utilizado para enviar a trama, enquanto que o módulo router foi ligado a outra porta.

O resultado do teste pode ser visto na **Figura 2.11**, abaixo, em que o terminal que enviou a trama se encontra do lado esquerdo e o que recebeu a trama está do lado direito. A informação está representada em hexadecimal, sendo que a azul estão os dados enviados e a vermelho os dados recebidos (do ponto de vista de cada módulo/terminal).

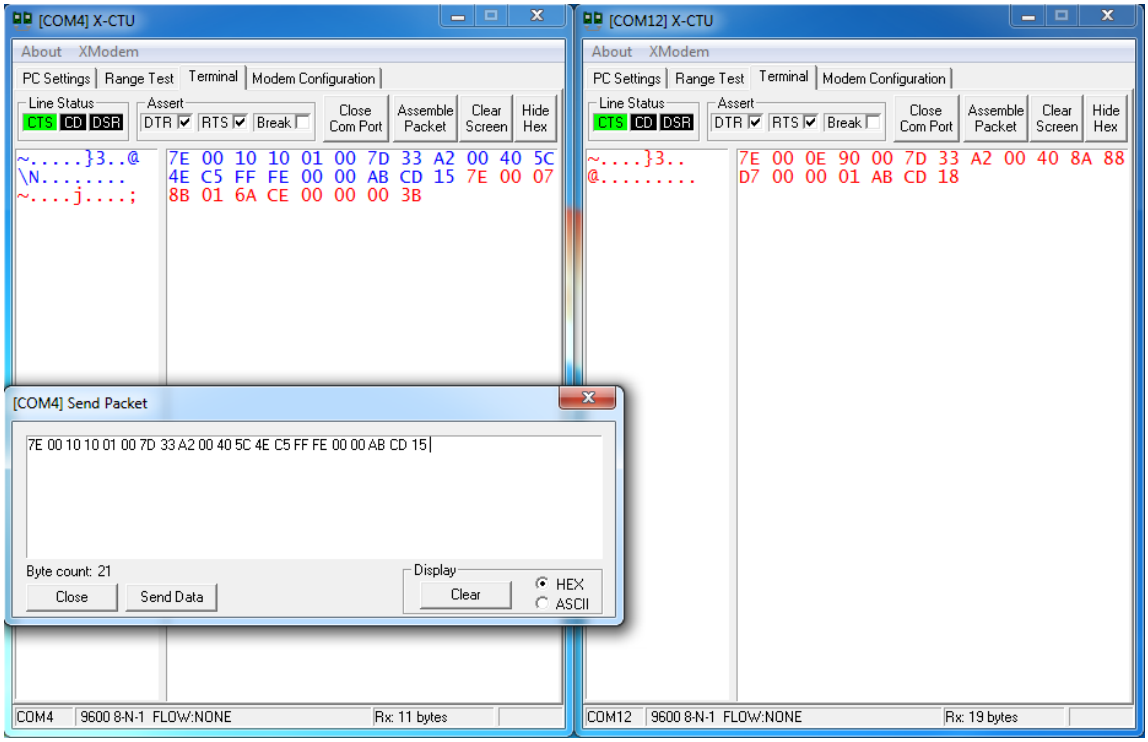



**Figura 2.11** – Tramas resultantes do envio de um pacote de dados entre dois módulos ligados ao PC.

Neste caso estão perfeitamente identificadas as três tramas envolvidas numa transmissão de dados. Note-se que estão representadas com alguns bytes ainda "mascarados". Procede-se à análise das tramas, já convertidas para o formato original.

Do lado esquerdo a azul está a trama enviada pelo módulo ligado à porta COM4.

| Start | Length | Frame type | Frame ID | 64bit Address (destination) | 16bit Address (destination) | Broadcast Radius | Options | RF Data | Checksum |
|---|---|---|---|---|---|---|---|---|---|
| 7E | 0010 | 10 | 01 | 0013A200405C4EC5 | FFFE | 00 | 00 | ABCD | 15 |

Como esta trama já foi analisada anteriormente em detalhe, agora são apenas referidos os campos mais relevantes: o campo do endereço de 64bit destino é preenchido com o endereço único do módulo router (ligado na porta COM12); o módulo destino é endereçado pelo seu endereço único, pelo que o campo de endereço de 16bit é preenchido com 0xFFFE; o campo de dados é preenchido com o valor hexadecimal 0xABCD.

Do lado direito a vermelho encontra-se a trama recebida pelo módulo *XBee* router ligado à porta COM12.

| Start | Length | Frame type | 64bit Address (sender) | 16bit Address (sender) | Receive Options | Received RF Data | Checksum |
|---|---|---|---|---|---|---|---|
| 7E | 000E | 90 | 0013A200408A88D7 | 0000 | 01 | ABCD | 18 |

Aqui também são mencionados apenas os campos relevantes. O valor no campo de endereço de 64bit indica que foi o módulo *XBee* ligado à porta COM4 a enviar o pacote, e o campo seguinte identifica esse mesmo módulo como o coordenador da PAN, pois possui o endereço de rede 0x0000. Os dados presentes no campo de dados coincidem com os que foram enviados pelo emissor, tal como seria de esperar.

Por fim, do lado direito a vermelho está a trama recebida pelo módulo *XBee* ligado à porta COM4 após o envio da trama a azul.

| Start | Length | Frame type | Frame ID | 16bit Address (destination) | Transmit Retry Count | Delivery Status | Discovery Status | Checksum |
|---|---|---|---|---|---|---|---|---|
| 7E | 0007 | 8B | 01 | 6ACE | 00 | 00 | 00 | 3B |

Nesta trama os campos mais relevantes de analisar são: no 3º campo, o valor 0x8B identifica a trama como sendo do tipo *ZigBee Transmit Status*; o campo de endereço de 16bit indica que o módulo *XBee* destino tem o endereço de rede 0x6ACE; o campo *Delivery Status*, com o valor 0x00, indica que o pacote foi correctamente recebido pelo destinatário.

Verifica-se assim a comunicação entre dois terminais do PC, através de dois módulos *ZigBee* devidamente configurados.

No caso de um emissor e um receptor, estão todas as combinações validadas e analisadas, quer através de um modo directo com o terminal, quer de uma maneira mais *ad hoc* utilizando os leds dos *Arduinos*.

**Teste 5. PC -> Dois *Arduinos***

No presente teste, pretendeu-se testar a capacidade de um emissor endereçar dois destinatários, quer directamente ao endereço de cada um, quer ambos simultaneamente (*broadcast*). Para tal utilizaram-se três módulos *XBee*: um ligado ao PC, um ao *Arduino Uno* e outro ao *Arduino Duemilanove*. Este teste consistiu no envio de tramas através do terminal da porta COM ligada ao *XBee* do PC. Os *Arduinos*, executando o mesmo código, estavam constantemente à espera de receber uma trama; quando recebiam, consoante a informação contida no campo de dados, piscavam o led de maneira pré-definida, permitindo ao utilizador identificar não só a recepção do pacote, mas também a informação transportada por ele.

Por forma a testar o cenário acima, do PC foram enviadas tramas através do terminal, construídas "à mão" com os dados que se pretendiam enviar. O código dos *Arduinos* consistia simplesmente em estar em ciclo infinito a tentar receber um novo pacote de dados e, quando recebia, verificava o seu tipo. Caso se tratasse de uma trama *ZigBee Receive Packet*, verificava o seu conteúdo: caso tivesse o valor 0xABCD, piscava o led 3 vezes; caso tivesse o valor 0xCDAB, piscava o led 5 vezes; caso não fosse nenhum dos valores anteriores, nada acontecia, voltando a esperar por um novo pacote.

Neste teste foi introduzido mais um módulo *XBee*, que foi ligado ao *Arduino Duemilanove*, de endereço único 0x0013A2004079606C. Este módulo foi configurado com o *software* X-CTU por forma a ficar em concordância com os restantes utilizados: mesmo valor da PAN ID (250), operando como ZNET 2.5 ROUTER/END DEVICE API, e parâmetro AP definido como 2 ("*escaped characters enables*"). Quanto aos restantes dois módulos, ligou-se o coordenador ao PC, enquanto que o outro foi ligado ao *Arduino Uno*.

Tal como no teste 3, apesar de não se analisarem directamente as tramas transmitidas, foi possível comprovar que a comunicação foi bem conseguida. Quando se enviava uma trama com o valor 0xABCD ou 0xCDAB para um dos endereços dos módulos da rede, o *Arduino* ligado ao módulo destino piscava o número de vezes correspondente ao valor enviado no campo de dados. Se não se endereçasse um módulo existente, ou o valor enviado na trama não fosse nenhum dos dois mencionados, nada acontecia. Além disto, no terminal do módulo emissor, cada vez que se enviava um pacote de dados, recebia-se imediatamente a trama com o estado do envio com a informação de que o pacote tinha sido correctamente recebido pelo receptor (ou não).

Também no caso de *broadcast*, a comunicação foi conseguida com sucesso. A principal diferença consistiu na construção da trama no emissor, em que o campo do endereço de 64bit do destinatário foi preenchido com o valor 0x000000000000FFFF (indicação para enviar para todos os elementos da rede). Neste cenário, como era de esperar, ambos os *Arduinos* piscam os leds simultaneamente e em concordância, pois recebem o pacotes iguais e ao mesmo tempo.

**Teste 6. PC -> Dois *Arduinos* (com retransmissão)**

Este teste avaliou a capacidade de enviar dados para um dispositivo da rede indirectamente, usando outro elemento como intermediário na transmissão. Neste cenário, todos os pacotes de dados foram enviados para o mesmo módulo *XBee*, indicando no

campo de dados o destinatário final do pacote. Tal como anteriormente, foram usados os leds dos *Arduinos* para interpretar a recepção dos pacotes de dados.

O cenário de teste montado consistiu no envio de tramas através do terminal no PC, endereçando sempre o módulo *XBee* conectado ao *Arduino Uno*, e incluindo no campo de dados a indicação se o pacote era destinado a este ou ao *Arduino Duemilanove*. Caso fosse enviado no campo de dados o valor 0xABCD, o *Arduino Uno* piscava o seu led 3 vezes; caso fosse enviado 0xCDAB, então reencaminhava o pacote de dados, endereçado ao módulo *XBee* conectado ao *Arduino Duemilanove*. Por sua vez, este executava o mesmo código que no teste anterior, limitando-se a receber tramas e piscar o led um número de vezes pré-definido consoante o conteúdo do campo de dados fosse 0xABCD ou 0xCDAB. Como o pacote de dados apenas chegava a si caso fosse enviado 0xCDAB pelo computador, o seu led piscava sempre 5 vezes.

Os módulos *XBee* e suas configurações mantiveram-se inalterados do teste anterior, mantendo-se também ligados aos mesmos elementos (PC e *Arduinos*), pelo que as únicas alterações são ao nível do *software* no *Arduino Uno* e na construção das tramas no PC.

A comunicação com ambos os *Arduinos*, directa ou indirectamente, foi bem conseguida. Mesmo não tendo acesso directo às tramas recebidas e trocadas entre os *Arduinos*, o correcto funcionamento foi comprovado pelo piscar dos leds de um de outro. O envio de uma trama com o valor 0xABCD no campo de dados fez o *Arduino Uno* piscar o seu led 3 vezes; o envio de uma trama com o valor 0xCDAB levou o *Arduino Duemilanove* a piscar o seu led 5 vezes.

Além do observado nos *Arduinos*, também no terminal surgiu, após o envio de uma trama, uma do tipo *ZigBee Transmit Status* com a informação de que o pacote foi recebido pelo destinatário (*Arduino Uno*) com sucesso.

**Teste 7. Dois *Arduinos* (simultaneamente) -> PC**

Este foi o teste que mais se aproximou do cenário da robótica cooperativa, pois temos dois *Arduinos* a enviar pacotes de dados para o PC simultaneamente a uma taxa transmissão elevada, o que se assemelha à situação em que temos mais de um veículo móvel a transmitir dados sensoriais para a estação central. Pretendia-se aumentar a taxa de envio de tramas para o módulo *XBee* no PC, por forma a encontrar o ponto de "quebra", isto é, a taxa de envio à qual o receptor começa a perder pacotes de dados.

Para concretizar o estudo exposto acima, os emissores (*Arduinos*) enviaram dados periodicamente e simultaneamente para a estação central. Como se pretendia testar a capacidade de comunicação independentemente dos dados enviados, estes consistiram simplesmente em valores inteiros por ordem: crescente no caso de uma fonte; decrescente na outra. Do lado dos *Arduinos*, o código consistiu no incremento em *loop* de uma variável de 0 até 10, e envio do valor actual a cada iteração. O *Arduino Uno* incrementava a variável, enquanto que o *Duemilavone* a decrementava. No PC, o *Matlab* limitou-se a receber e representar os dados.

Como neste teste a análise de uma trama no terminal não era suficiente, foi implementada em *Matlab* uma rotina para recepção, interpretação e representação gráfica dos dados recebidos no PC.

A periodicidade do envio dos dados pelo *Arduino* depende essencialmente de três factores: o tempo de execução do código para preparar os dados; o tempo que demora a biblioteca e o módulo *XBee* a criar a trama e a enviá-la, que no *Arduino Uno* demora cerca de 20 ms; e finalmente o *delay* introduzido propositadamente em cada iteração dentro do *loop*. Este *delay* foi sucessivamente reduzido, impondo uma taxa de transmissão cada vez maior. No PC, mais concretamente no *Matlab*, o teste foi corrido não em termos de tempo mas em número total de pacotes recebidos, pelo que se os *Arduinos* tiverem códigos estruturalmente idênticos, tempos de atraso iguais e velocidades de processamento aproximadamente iguais, então o PC deve receber o mesmo número de pacotes das duas fontes.

Na fase final deste teste, com taxas já muito elevadas, o mesmo teste foi executado primeiramente com cada *Arduino* individualmente, e apenas depois com ambos a enviar simultaneamente.

Teste 1 – Envio simultâneo a 1Hz

- 40 Pacotes recebidos
- *Delay* de 1 segundo no *Arduino*, mas é preciso ter em conta que na recepção das tramas de estado do envio utilizam-se os leds para sinalizar o resultado, o que introduziu mais atrasos na execução do código.
- Neste teste o valor a ser enviado estava no intervalo de 1 a 50.

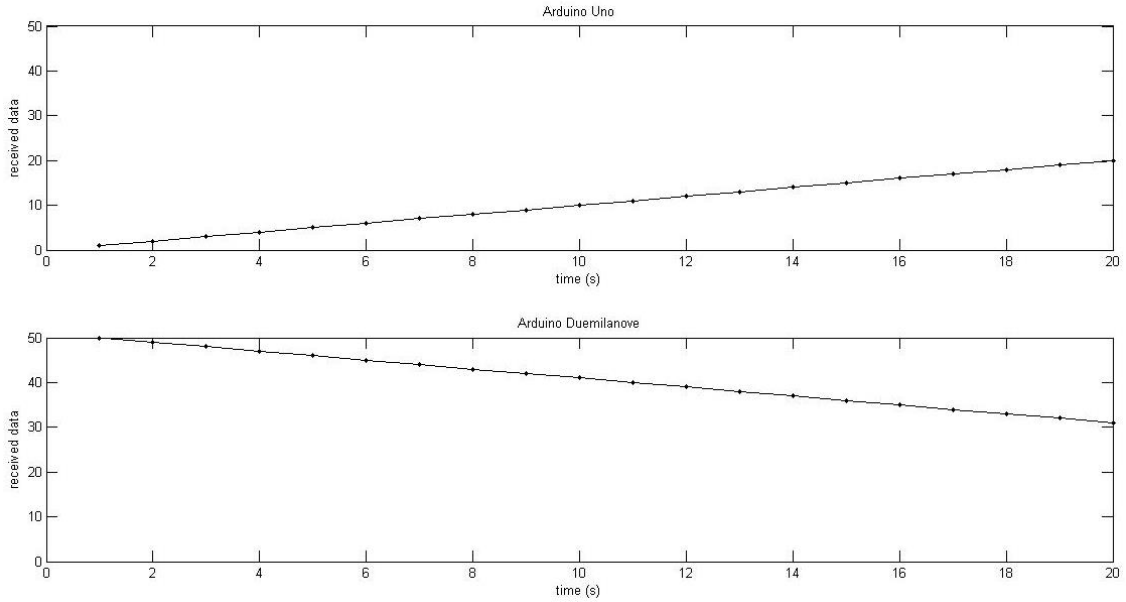



**Figura 2.12** – Dados resultantes do Teste 1.

Neste primeiro caso verifica-se um funcionamento correcto, com os dados dos *Arduinos* recebidos a cada segundo, não havendo qualquer perda de pacotes. O tempo de teste foi de cerca de 20 segundos.

Teste 2 – Envio simultâneo a 2Hz

- 40 Pacotes recebidos
- Atraso de 500 milissegundos, também com a utilização dos leds.

- Aqui a variável incrementada variava de 0 a 9, enquanto que a decrementada de 10 a 0.

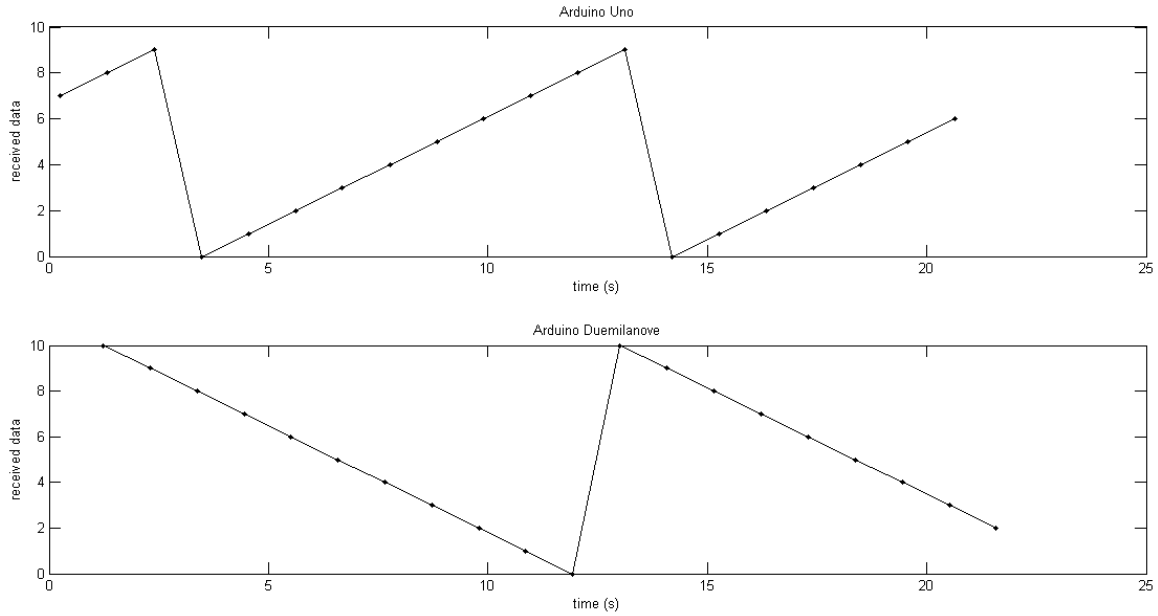



**Figura 2.13** – Dados resultantes do Teste 2.

Novamente não há qualquer perda de pacotes, decorrendo o teste durante aproximadamente 21,5 segundos. Note-se que aqui a sinalização da troca de pacotes usando os leds começa a introduzir atrasos significativos nas comunicações, razão pela qual a taxa de envio é reduzida para cerca de metade da desejada, ou seja, neste período de 21,5 segundos são recebidos 20 pacotes, em vez dos 43 que seriam esperados à taxa de 2Hz.

Teste 3 – Envio simultâneo a 4Hz

- 40 Pacotes recebidos
- *Delay* de 250 milissegundos, mais uma vez com os leds.
- A gama das variáveis é a mesma que no teste anterior.

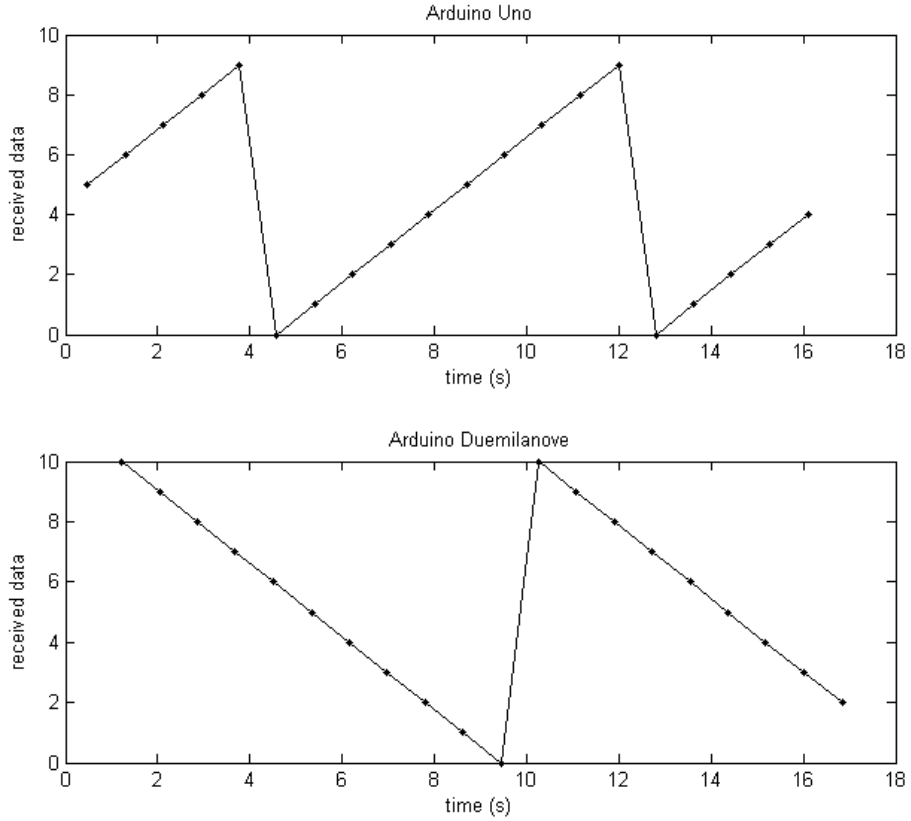



**Figura 2.14** – Dados resultantes do Teste 3.

O correcto funcionamento do sistema é assegurado. Aqui, como no teste anterior, o tempo total do teste é superior ao esperado devido aos atrasos acrescidos pela utilização dos leds. Por exemplo neste caso, partindo do princípio que são recebidos 20 pacotes provenientes de cada *Arduino*, então seria de esperar que demorasse 5 segundos, o que na realidade se traduz em cerca de 17 segundos. A diferença é muito elevada devido à utilização dos leds na sinalização da troca de pacotes, pois têm *delays* associados da ordem das dezenas de milissegundos, o que não é desprezável.

Teste 4 – Envio simultâneo a 10Hz

- 100 Pacotes recebidos
- *Delay* de 100ms, mas finalmente sem os atrasos relacionados com o uso dos leds na sinalização da comunicação.

Neste teste, por se tratar de uma taxa de transmissão mais elevada, correram-se os mesmos testes primeiro com cada *Arduino* individualmente e só depois simultaneamente. Desta forma foi mais fácil comparar o desempenho e verificar se houve perdas de velocidade de transmissão ou mesmo de pacotes.

- Apenas *Arduino Duemilanove*:

Após o momento em que a porta série é aberta e o programa no *Matlab* começa a ler os dados recebidos, o primeiro pacote chega no instante t=1,237seg. Não se verifica qualquer perda de dados, que chegam com um período que varia entre 0,126seg e 0,177seg, com média de 0,151seg.

- Apenas *Arduino Uno*:

O *Arduino Uno* começa a enviar pacotes praticamente de imediato quando a porta série é aberto no *Matlab* (t=0,032seg). O período de recepção de pacotes no *Matlab* varia entre 0,109seg e 0,142seg, tendo período médio de 0,122seg e não havendo perdas de pacotes.

- Ambos *Arduinos* simultaneamente:

Tal como aconteceu quando transmitiu apenas o *Arduino Duemilanove*, também aqui foi recebido o seu primeiro pacote no instante t=1,22seg, pelo que o nesse tempo inicial apenas o *Arduino Uno* transmitiu pacotes. Isto implica que o número total de pacotes recebidos pelo *Matlab* não está igualmente distribuído. Os resultados são apresentados na **Tabela 2.9** e **Figura 2.15**.

**Tabela 2.9** – Estatísticas dos envios de dados pelos *Arduinos*, no Teste 4.

| | Período (s) | | | Freq (pacotes/s) | Nº pacotes enviados |
|---|---|---|---|---|---|
| | Min | Max | Med | | |
| ***Duemilanove*** | 0,132 | 0,175 | 0,151 | 6,63 | 40 |
| ***Uno*** | 0,096 | 0,163 | 0,122 | 8,2 | 60 |

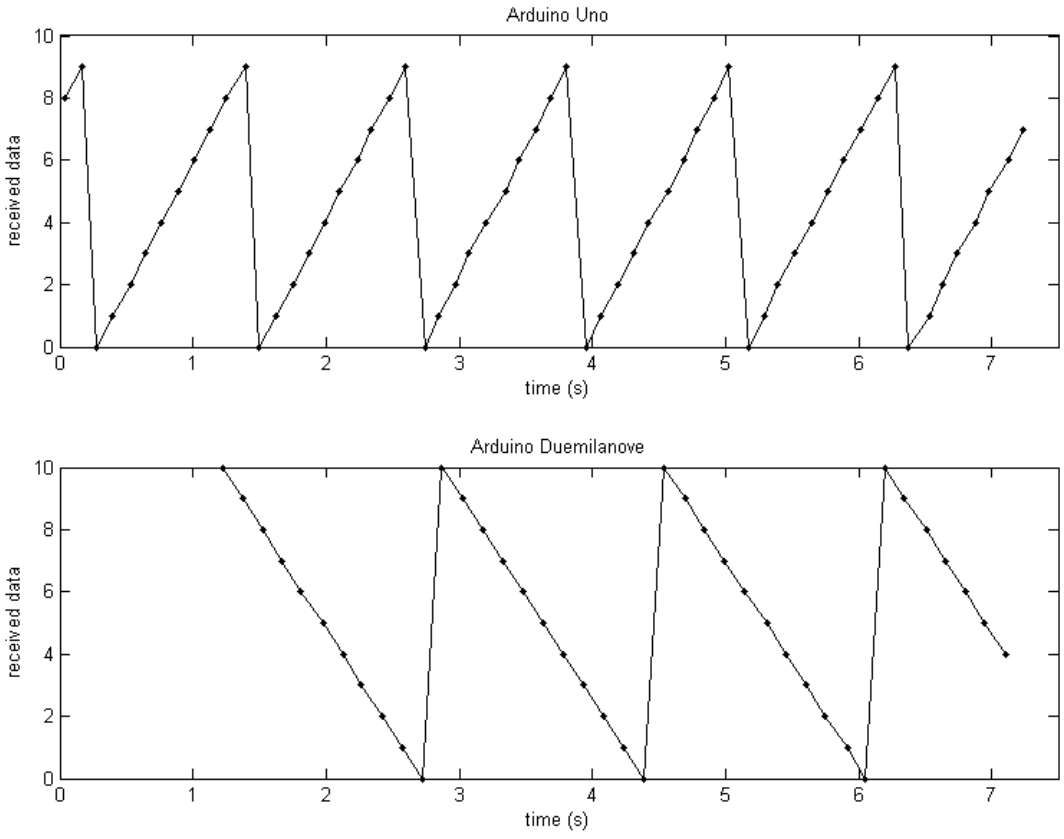



**Figura 2.15** – Dados resultantes do Teste 4.

Comparando os períodos médios de envio de tramas de cada *Arduino* isoladamente com os períodos médios de envio quando ambos transmitem simultaneamente, verifica-se que não existe perda de desempenho do primeiro para o segundo caso. Analisando a **Figura**

**2.15**, observa-se o tempo inicial em que o *Arduino Duemilanove* não envia qualquer pacote, enviando apenas o *Arduino Uno*. Confirma-se ainda que não há perdas de tramas.

Idealmente, o período deveria ser de 100ms, pois é esse o atraso introduzido entre cada cálculo e envio da variável. Porém, o atraso associado ao envio de um pacote de dados *XBee* é de cerca de 20ms, o que aproxima o período de envio do *Arduino Uno* do valor real, pois está muito próximo dos 120ms teóricos. Este atraso de 20ms associado à construção e envio de um pacote de dados vai ser observado repetidamente nos testes seguintes.

Na **Tabela 2.9** verifica-se que o *Arduino Uno* enviou mais pacotes que o *Duemilanove*. Isto deve-se ao instante inicial em que apenas o *Uno* envia dados, e a ligeiras diferenças entre os *Arduinos*.

Teste 5 – Envio simultâneo a 20Hz

- 100 Pacotes recebidos
- *Delay* de 50ms, sem utilização dos leds.

Tal como foi feito no teste anterior, também aqui se correram os testes com os *Arduinos* de forma individual, colocando posteriormente os dois a correr ao mesmo tempo. Os resultados individuais foram os seguintes.

- *Arduino Duemilanove*

O primeiro pacote recebido pelo *Matlab* surge no instante t=1,223seg, sendo os seguintes recebidos com período a variar entre 0,086seg e 0,122seg, com média de 0,101seg.

- *Arduino Uno*

O *Arduino Uno*, ao contrário do *Duemilanove*, começa a enviar pacotes de imediato. A periodicidade deste envio varia de 0,05seg até 0,098seg e tem valor médio de 0,072seg.

- Ambos simultaneamente

Finalmente foram colocadas as duas fontes a enviar dados para o *Matlab* e traçados os respectivos gráficos com a informação temporal. Analogamente ao constatado anteriormente, quer individual ou simultaneamente, o *Arduino Duemilanove* pára no início da comunicação, enviando apenas o *Arduino Uno* durante os 1,21 segundos iniciais. Os períodos de transmissão constam na **Tabela 2.10**, e os dados recebidos ao longo do tempo estão na **Figura 2.16**.

**Tabela 2.10** – Estatísticas dos envios de dados pelos *Arduinos*, no Teste 5.

| | Período (s) | | | Freq (pacotes/s) | Nº pacotes enviados |
|---|---|---|---|---|---|
| | Min | Max | Med | | |
| ***Duemilanove*** | 0,079 | 0,145 | 0,11 | 9,06 | 33 |
| ***Uno*** | 0,041 | 0,099 | 0,072 | 13,97 | 67 |

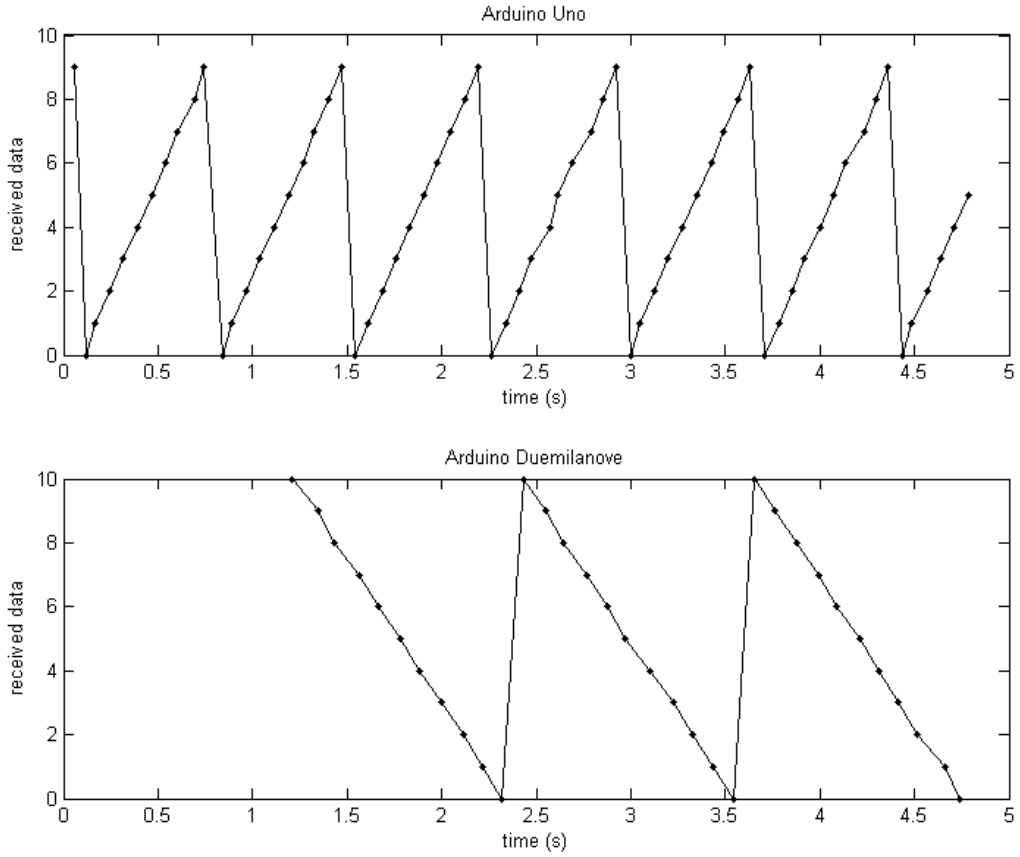



**Figura 2.16** – Dados resultantes do Teste 5.

Pela análise dos períodos de envio nos testes individuais, e pelos que constam na **Tabela 2.10**, comprova-se que não existe degradação dos tempos médios de transmissão de tramas quando ambos os *Arduinos* transmitem em simultâneo. Verifica-se ainda que o período de envio de dados do *Arduino Uno* se aproxima do esperado, se considerarmos que o atraso associado à construção e envio da trama são os já mencionados 20ms. Somando este atraso com o *delay* de 50ms propositadamente introduzido no código, o período de envio resultante é muito próximo dos 72ms medidos na prática.

Observando a **Figura 2.16**, verifica-se que não existe qualquer perda de pacote. No entanto, a recepção dos mesmos não é linear, havendo de vez em quando pequenas variações no tempo que provêm da recepção quase coincidente no tempo de ambas as fontes de dados.

Teste 6 – Envio simultâneo a 100Hz

- 100 Pacotes recebidos
- *Delay* de 10ms, sem leds.
- Ambas as variáveis aqui variam entre 0 e 10.

Seguem-se então os resultados de cada *Arduino* nos testes individuais e conjunto.

- *Arduino Duemilanove*

O *Matlab* recebe o primeiro pacote proveniente do *Arduino* no instante t=1,245seg. Os seguintes chegam com um período médio de 0,061seg, variando entre 0,018seg e 0,084seg.

- *Arduino Uno*

Neste caso os pacotes começam a ser recebidos imediatamente, com período médio de 0,032seg variando entre 0,010seg e 0,06seg.

- Ambos

Quando colocados os *Arduinos* a enviar simultaneamente, surge o cenário já anteriormente visto, em que apenas o *Uno* envia até ao instante t=1,212seg, momento em que é recebido o primeiro pacote proveniente do *Duemilanove*. Isto justifica o facto de serem recebidos muito mais dados provenientes do *Arduino Uno*.

**Tabela 2.11** – Estatísticas dos envios de dados pelos *Arduinos*, no Teste 6.

| | Período (s) | | | Freq (pacotes/s) | Nº pacotes enviados |
|---|---|---|---|---|---|
| | Min | Max | Med | | |
| ***Duemilanove*** | 0,056 | 0,101 | 0,075 | 13,35 | 19 |
| ***Uno*** | 0,01 | 0,055 | 0,031 | 31,95 | 81 |

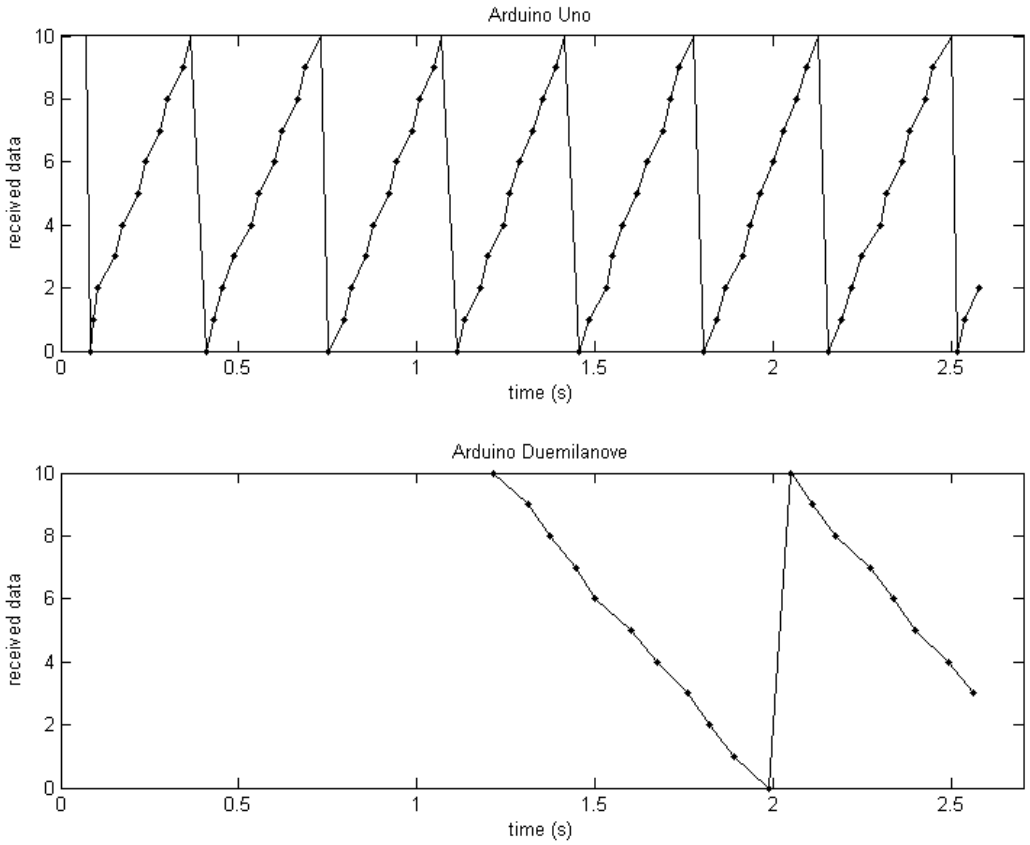



**Figura 2.17** – Dados resultantes do Teste 6.

Comparando novamente as taxas de envio nos testes individual e simultâneo, observa-se que não há degradação das taxas de transmissão das tramas quando são

colocados os *Arduinos* a transmitir simultaneamente. Mantem-se o atraso observado no início da transmissão de tramas por parte do *Duemilanove*.

É de salientar o facto que o período de recepção de pacotes provenientes do *Arduino Uno* continua próximo do previsto, pois a soma do *delay* de 10ms propositadamente introduzido com o atraso de 20ms associado ao envio de um pacote dá um período teórico de 30ms, que é próximo dos 31ms que se verificam na prática.

Na **Figura 2.17** observa-se já uma grande irregularidade na periodicidade de recepção dos dados. A maneira como a variável varia no código do *Arduino* origina uma função do tipo "dente de serra", e nesta fase a forma dessa onda já não é tão linear como era inicialmente. Porém não se verificam ainda quaisquer perdas de dados.

Teste 7 – Envio de Tramas à taxa máxima

- 1000 Pacotes recebidos
- Sem *delay* introduzido no código, portanto a taxa de transmissão de pacotes depende apenas do tempo de processamento das instruções e do tempo de construção e envio dos pacotes pelos módulos *XBee*.

Neste teste final, pôs-se à prova a capacidade das comunicações com a taxa máxima de envio de tramas que o *hardware* utilizado é capaz de produzir. Retirou-se o *delay* introduzido propositadamente no código do *Arduino*, fazendo com que a taxa máxima de transmissão dependa apenas da velocidade de processamento do *Arduino* e da geração e envio de cada trama. Mais uma vez, os testes foram corridos com cada Arduino individualmente antes de os pôr a enviar simultaneamente. Aumentou-se também o número total de pacotes a serem recebidos no *Matlab*. Os resultados são apresentados a seguir.

- *Arduino Duemilanove*

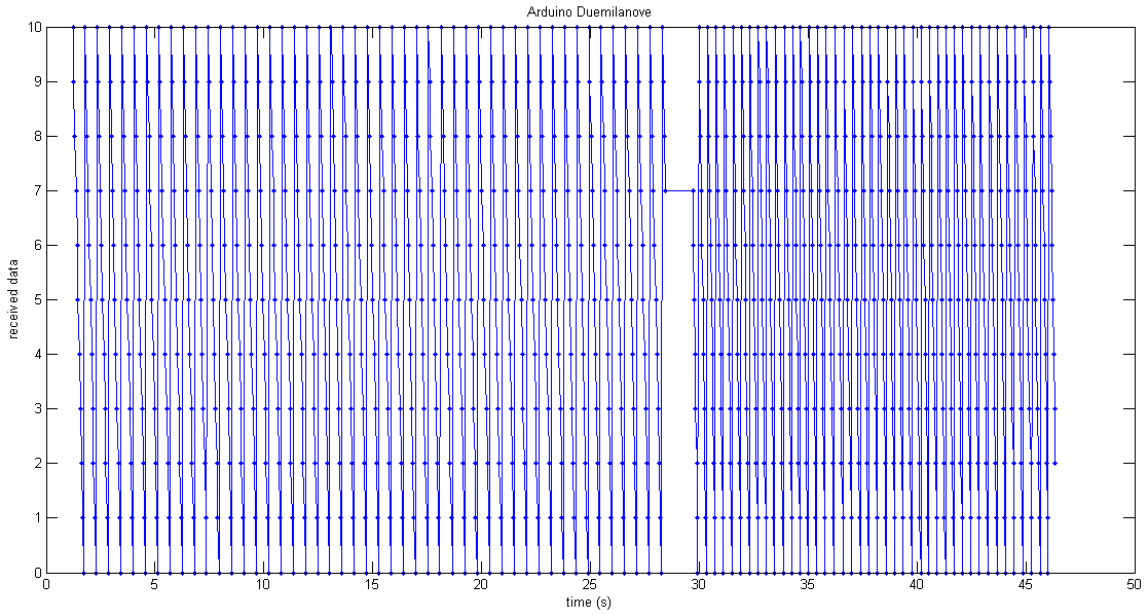



**Figura 2.18** – Resultado do envio periódico, sem *delays*, de dados apenas pelo *Arduino Duemilanove*.

Do gráfico acima observa-se o tempo inicial em que o *Arduino* não envia qualquer pacote, começando a enviar a partir do instante t=1,23seg. Depois disto, são recebidas sem

falhas todas as tramas, com periodicidade média de 45ms. O que salta mais à vista no gráfico é a falha de comunicação temporária perto dos 27 segundos, com a duração de 1,25 segundos. Durante este tempo não existe qualquer troca de tramas (visível pela ausência de sinalização dos próprios módulos *XBee*). Porém, após esta falha, a comunicação é reestabelecida, e é inclusive feita a uma taxa de transmissão superior. À parte da falha de comunicação, não foram registadas perdas de pacotes.

- *Arduino Uno*

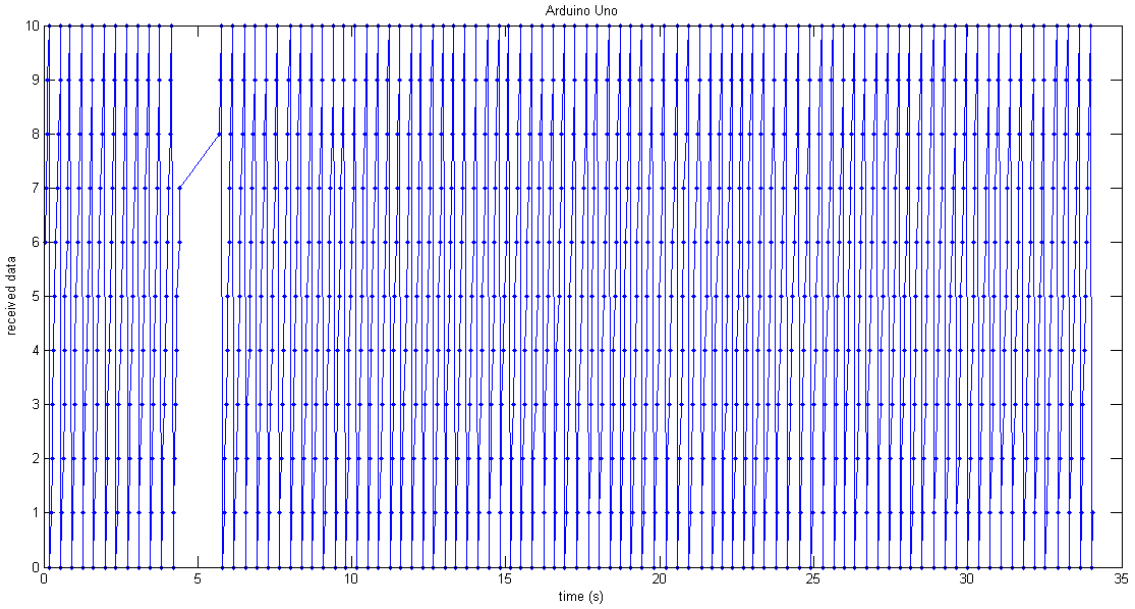



**Figura 2.19** – Resultado do envio periódico, sem *delays*, de dados apenas pelo *Arduino Uno*.

O *Arduino Uno* começa a enviar as tramas de imediato. Também aqui se verifica uma falha de comunicação por volta dos 4 segundos, com duração de 1,31 segundos. Após a falha, a comunicação é reposta e a recepção de pacotes é feita com a mesma periodicidade média de 34,1ms (sendo o mínimo de 8,5ms), incluindo o tempo sem comunicação. Também aqui não são verificadas perdas de dados.

- Ambos

Finalmente, com ambos os *Arduinos* a enviar dados simultaneamente, presenciam-se os mesmos fenómenos que nos testes individuais: apenas o *Arduino Uno* envia pacotes durante os primeiros 1,23 segundos, e existe uma falha de comunicação entre todos os elementos da rede (PC e ambos os *Arduinos*). Esta falha tem duração de cerca de 1,27 segundos, após a qual o *Arduino Duemilanove* aumenta a taxa de envio, tal como acontecera no teste individual. Os resultados são apresentados na **Tabela 2.12** e **Figura 2.20**.

**Tabela 2.12** – Estatísticas dos envios de dados pelos *Arduinos*, no Teste 7.

| | Período (s) | | | Freq (pacotes/s) | Nº pacotes enviados |
|---|---|---|---|---|---|
| | Min | Max | Med | | |
| ***Duemilanove*** | 0,0089 | 1,282 | 0,055 | 18,15 | 403 |
| ***Uno*** | 0,0085 | 1,269 | 0,039 | 25,45 | 595 |

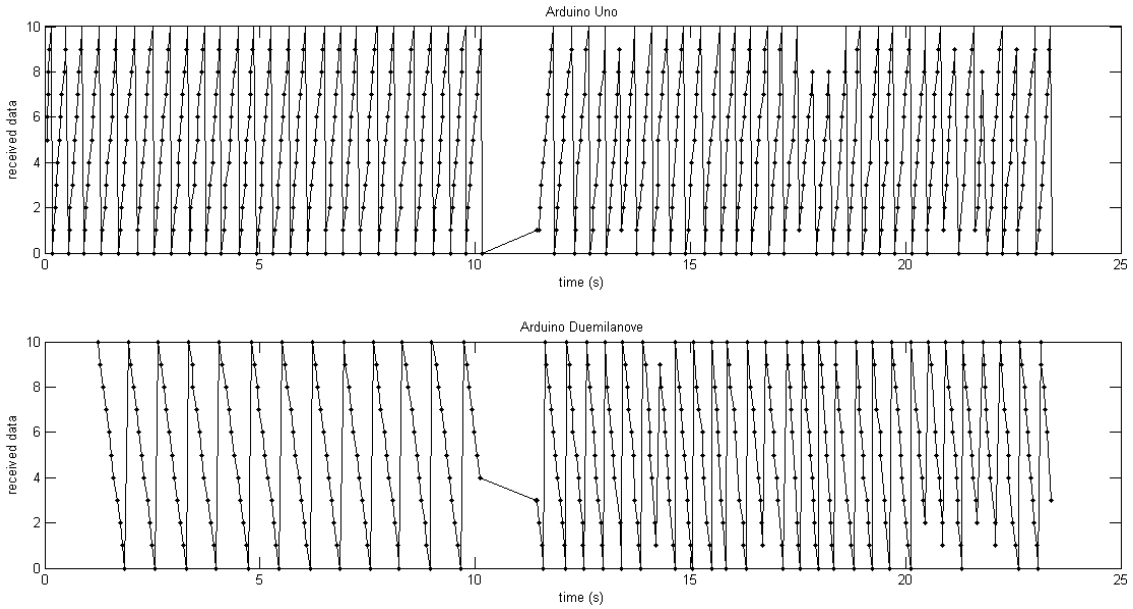



**Figura 2.20** – Resultados do envio simultâneo de dados, sem *delays*, por ambos os *Arduinos*.

Da análise dos tempos verifica-se que não há redução da taxa de transmissão dos *Arduinos* no caso de simultaneidade quando comparado com o caso em que funcionam individualmente. Note-se que apesar de se diminuir o *delay* introduzido no código do *Arduino*, o *Uno* parece ter um período maior do que no teste anterior, em que tem *delay* (acrescido) de 10ms. Isto deve-se ao facto de o tempo sem comunicação também ser contabilizado para o cálculo do período médio.

Analisando agora a **Figura 2.20**, têm-se dois momentos na transmissão: antes da falha de comunicação, não se regista qualquer perda de pacotes; após a falha, o *Arduino Duemilanove* aumenta ligeiramente a sua taxa de envio de dados e começam a notar-se algumas perdas de informação das duas fontes de dados. Isto nota-se nos gráficos da **Figura 2.20** pela falta de alguns valores nas formas das funções "dente de serra".

Conclui-se portanto que após a falha de comunicação, os emissores ajustam as suas taxas de envio de dados para o máximo, sobrecarregando o receptor no PC, e causando perdas de pacotes provenientes de ambas as fontes.

## 2.7. Observações Finais

Neste capítulo foi introduzido e explicado o protocolo de comunicação sem fios *ZigBee* e foram apresentadas algumas das tramas que ele suporta.

Foi descrito um conjunto de cenários de teste ao funcionamento e capacidades dos módulos *XBee* na interligação de vários dispositivos, nomeadamente microcontroladores e PC. Os testes tiveram em conta os vários sentidos de comunicação possíveis e permitiram uma avaliação preliminar da velocidade de transmissão de dados que pode ser atingida.

Os resultados obtidos foram analisados em detalhe e reforçam a convicção de que o protocolo de comunicação *ZigBee*, bem como os rádios *XBee* utilizados no projecto, satisfazem os requisitos de comunicação do sistema robótico a desenvolver.

# 3. Sensores e Actuadores

## 3.1. Introdução

Os sensores num *robot* podem ser considerados os seus sentidos. Os actuadores, por sua vez, podem ser vistos como os seus músculos, possibilitando movimento.

Os sensores permitem traduzir acontecimentos no mundo físico para sinais eléctricos interpretáveis por microcontroladores ou outros dispositivos electrónicos **[16]**. Os actuadores são os elementos que permitem aos *robots* (ou outros equipamentos automatizados) interagir com o mundo físico.

Actualmente, praticamente todas as áreas, da indústria à medicina, estão até algum nível automatizadas. Podem encontrar-se sensores e actuadores em aplicações robóticas nos mais variados contextos, como a indústria automóvel ou sistemas de segurança. Desenvolveu-se por isso uma grande variedade de sensores e actuadores, para satisfazer os requisitos específicos dos sistemas onde são aplicados.

Podem indicar-se algumas categorias de sensores, como de temperatura, de proximidade ou de toque. Na robótica, os mais utilizados são os sensores de distância, odométricos e acelerómetros (ou giroscópios).

Os actuadores estão presentes em braços robóticos e outros manipuladores, bem como nos sistemas de locomoção de *robots*. Existem já actuadores que tendem a imitar os músculos humanos, tais como os músculos artificiais pneumáticos ou os fios musculares. Recentemente, tem vindo a desenvolver-se a tecnologia dos nanotubos elásticos, que são músculos artificiais que possuem grande capacidade de deformação elástica. Embora ainda em fase inicial, pode prever-se que esta tecnologia virá a permitir aos *robots* humanóides ultrapassarem os seres humanos em corrida ou mesmo em salto **[17]**.

## 3.2. Descrição do Hardware utilizado

O veículo robótico desenvolvido neste trabalho está munido de quatro tipos de sensores e dois tipos de actuadores. Os sensores são:

- Um sensor de distância de ultra-sons, destinado a detectar obstáculos a média/longa distância (desde 40cm a alguns metros). Os sensores deste tipo têm uma taxa de amostragem baixa (até pouco mais de uma dezena de amostras por segundo) mas são razoavelmente fiáveis e lineares. As suas características (tempo de resposta, gama de distâncias, forma do feixe sonoro do emissor, preço,…) podem variar consideravelmente. A forma do feixe sonoro – vide **Figura 3.1** – é um aspecto muito importante, pois os lobos laterais podem introduzir erro nas medidas.

- Um sensor de infravermelhos, destinado a detectar objectos a curtas distâncias (até 30cm). Os sensores deste tipo são mais precisos neste alcance que os de ultra-sons e têm uma taxa de amostragem superior (algumas dezenas de amostras por segundo). Porém, têm uma resposta não linear. Os sensores de infravermelhos existentes no mercado diferem entre si pela gama de distâncias a que funcionam, tempos de resposta, preço, etc. Podem ser digitais, activando uma saída quando detectam um obstáculo, ou analógicos, fornecendo um valor dependente da distância medida. Destes últimos, podem indicar-se, a título de exemplo, o SHARP GP2Y0A02YK0F, que funciona para distâncias entre 20cm e 150cm, cuja saída analógica de tensão é directamente proporcional à distância ao objecto, ou o SHARP GP2D12, eficaz no intervalo dos 10cm aos 80cm, e gera uma saída também analógica em tensão dependente da distância.
- Uma bússola digital, que permite obter a orientação global do veículo (no plano horizontal), e tem uma taxa de amostragem muito elevada (mais de uma centena de amostras por segundo). Porém, existem bússolas de dois eixos (no plano horizontal), tendo as mais sofisticadas compensação da inclinação e outras, ainda mais avançadas, sensíveis à orientação nos três eixos.
- Um *encoder* acoplado a cada uma das duas rodas do veículo, o que possibilita a contagem das voltas de cada roda (e cálculo do respectivo deslocamento com razoável precisão).

Os actuadores aplicados são:

- Um motor DC acoplado a cada uma das rodas motrizes do veículo. De facto, os actuadores mais comuns são motores, que podem ser alimentados por diversas fontes de energia e, na maior parte dos casos, a locomoção de *robots* é assegurada por motores eléctricos.
- Um motor servo ao qual são acoplados ambos os sensores de distância no mesmo eixo vertical, de modo a poderem ser direccionados com precisão.

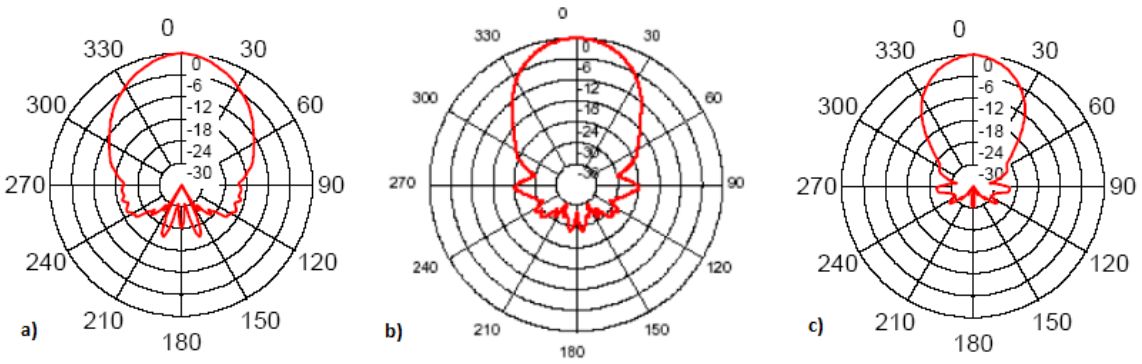



**Figura 3.1** – Feixes de Sonares: **a)** SRF10; **b)** SRF08; **c)** SRF01. O "campo de visão" de um sensor de ultra-sons depende da forma do feixe.

Além dos sensores e actuadores, o veículo robótico é ainda equipado com um módulo *XBee* para comunicar com os restantes elementos da rede de comunicações (PC e outros veículos robóticos), e também com um *Arduino*, que comanda todos os componentes do *robot*.

## 3.3. Sensores

### 3.3.1. Sensor de Distância de Ultra-sons

Um sensor de distância do tipo ultra-sons consiste num par de emissor e receptor colocados lado a lado segundo o mesmo sentido e orientação. O princípio de funcionamento baseia-se no tempo de voo[4]. O emissor envia um feixe sonoro, que é reflectido num obstáculo e é detectado pelo receptor. Conhecendo o tempo $T$ decorrido entre a emissão e recepção do feixe sonoro e a velocidade $v$ do som no meio, é possível calcular a distância $d$ ao obstáculo reflector **[16]**, segundo a seguinte fórmula:

$$d = \frac{T \times v}{2} \qquad 3.1$$

Apesar do princípio do sonar ser bastante simples, na verdade não é assim tão elementar, pois existem algumas situações que introduzem erro, mas que são explicadas pelas leis físicas da reflexão (sonora). Em **[18]** é feito um estudo mais aprofundado acerca destes fenómenos de reflexão do som e as suas implicações num sonar *range finder*. A principal causa de erro nos sensores ultra-sons é o *multipath*. Isto traduz-se numa situação em que o feixe reflectido recebido pelo sensor sofreu mais de uma reflexão, percorrendo um caminho mais longo do que a distância real ao obstáculo (e retorno). Posto isto, as principais fontes de erro são os cantos, principalmente os convexos. A **Figura 3.2** ilustra bem esta situação. A tracejado estão as paredes físicas do cenário, e os pontos ligados são as diferentes leituras de um sonar. Note-se que nos cantos, todos as leituras estão igualmente distanciadas do sensor.

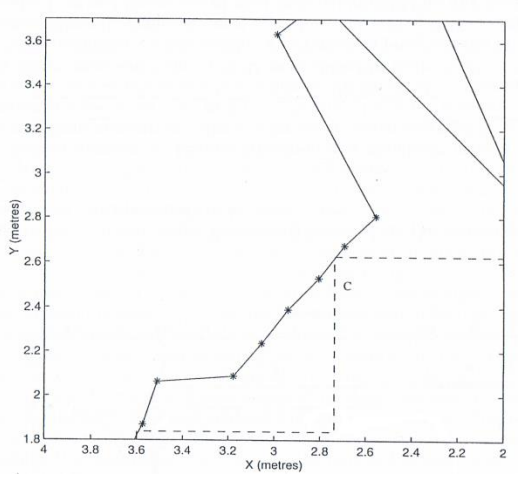



**Figura 3.2** – Efeitos do *multipath* em sensores de ultra-sons **[18]**.

[4] Traduzido do inglês "*time-of-flight*".

Após a emissão do feixe sonoro, o seu retorno é detectado quando a amplitude do sinal recebido ultrapassa um determinado *threshold*. Este *threshold* vai diminuindo ao longo tempo, pois sinal emitido vai perdendo amplitude ao longo do tempo. A amplitude da onda sonora reflectida depende também do material do obstáculo. Este, conjugado com o comprimento de onda do feixe emitido pelo sonar, faz com que a reflecção especular seja predominante. Isto significa que pode-se comparar o que um sonar "vê", como se estivéssemos a olhar para uma sala cheia de espelhos **[18]**. A

Um trabalho fundamental sobre a teoria de sonares *range finder* é apresentado por Kuc e Siegel em **[19]**. Aqui é feito um estudo aprofundado sobre as respostas de um sonar em situações peculiares, como cantos côncavos e convexos, ou em paredes perpendiculares ao plano de incidência do sensor. Neste segundo caso é a abertura do feixe sonoro que introduz erro. Estas situações estão bem ilustradas na **Figura 3.3**.

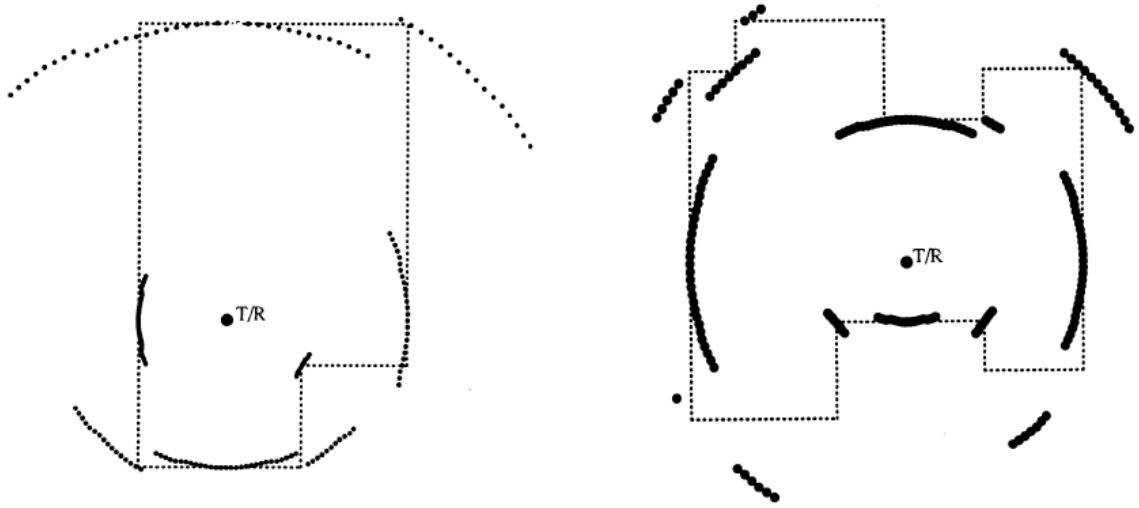



**Figura 3.3** – Resposta de um sonar *range finder* em dois cenários diferentes, em que T/R é a posição do Transmissor/Receptor. São visíveis os efeitos nos cantos, bem como as incidências perpendiculares **[19]**.

Em **[20]**, Cao e Borenstein estudam as respostas de um e de dois sonares em simultâneo quando orientados perpendicularmente a uma superfície especular. Eles estavam interessados em estudar a influência da abertura do feixe, bem como os lobos laterais bem conhecidos deste tipo de sonares. Com a utilização de dois sonares em conjunto, conseguiram reduzir consideravelmente os lobos laterais e fechar muito mais a abertura do feixe, melhorando a resolução angular do sensor (conjunto).

**Figura 3.4** – Sensor de distância de ultra-sons utilizado, SRF10 da Devantech (fonte: http://www.acroname.com/robotics/parts/R241-SRF10.html).

No presente trabalho, o sensor usado é o "*SRF10 Ultrasonic range finder*" da Devantech. A imagem do sensor está na **Figura 3.4** e tem 4 pinos: alimentação a 5V, *ground*, e os pinos SDA e SCL, correspondentes aos sinais de dados e *clock*.

A comunicação com este sensor é feita através de um barramento I2C, usando os pinos SDA e SCL. Neste tipo de comunicação, usa-se um sinal para sincronização entre os dois extremos da ligação (SCL) e um sinal para transmitir os dados (SDA). Por definição, o endereço do sensor é 0xE0; porém pode ser alterado para um de 15 outros endereços permitidos, o que significa que é possível ter até 16 sonares diferentes ligados ao mesmo microcontrolador.

O sensor SRF10 tem 4 registos internos com os quais são efectuadas as trocas de dados, quer para efeitos de configuração, quer para a obtenção de medições de distância. Uma medição de distância tem a duração de 65 milissegundos, desde que se requisita uma medição até à leitura da mesma. As medições podem ser efectuadas em: centímetros, polegadas ou tempo de voo (ida e volta) em microssegundos. O sonar consegue detectar objectos até 6 metros eficazmente.

A forma do feixe emitido pelo sonar não pode ser alterada e tem o formato representado na **Figura 3.1 a)**. Porém, não alterando as características específicas do sonar, podem utilizar-se truques simples que permitem reduzir os *side lobes* e tornam o feixe mais estreito. Esta solução está apresentada e explicada na secção 3.7.2.

Este sensor está localizado na zona frontal do veículo, afixado ao motor servo, conferindo-lhe um grau de liberdade: rotação horizontal (neste caso de -90° a +90°, relativamente à frontal).

Ao nível da programação, de modo a ter maior precisão nas medidas, estas são adquiridas pelo sonar como tempo de voo em microssegundos. Para filtrar o erro associado às medidas, no *Arduino* são adquiridas 5 amostras, com intervalos de tempo de 70ms entre si, e é feita uma média uniforme entre os 5 valores, que é enviada para o PC. A taxa de amostragem no PC é portanto aproximadamente 2,85 amostras/segundo.

Os testes efectuados a este componente foram muito simples. O sonar foi colocado estático frontal a uma superfície plana, e foram enviados os dados sensoriais adquiridos no *Arduino* para o PC, processados e representados graficamente no *Matlab*. A vantagem em utilizar medidas de tempo de voo em vez de centímetros é o facto de que no segundo caso tem-se resolução de 1cm, enquanto que em tempo de voo a resolução é superior, inferior ao

milímetro. A velocidade do som considerada para obter a distância em função do tempo de voo é 340m/s.

Apresentam-se 3 cenários exemplo (ver **Figura 3.5**) em que o sensor ultra-sons foi testado: uma primeira situação em que sensor foi colocado a cerca de 20cm de distância de uma superfície branca de acrílico; um segundo caso com o sonar colocado à mesma distância de uma placa de madeira forrada com pano; finalmente um terceiro cenário com o sensor apontado verticalmente para o tecto. Em nenhum dos cenários abordados foi aplicado qualquer tipo de filtro; estas são as medidas tal como são lidas do sensor pelo *Arduino*.

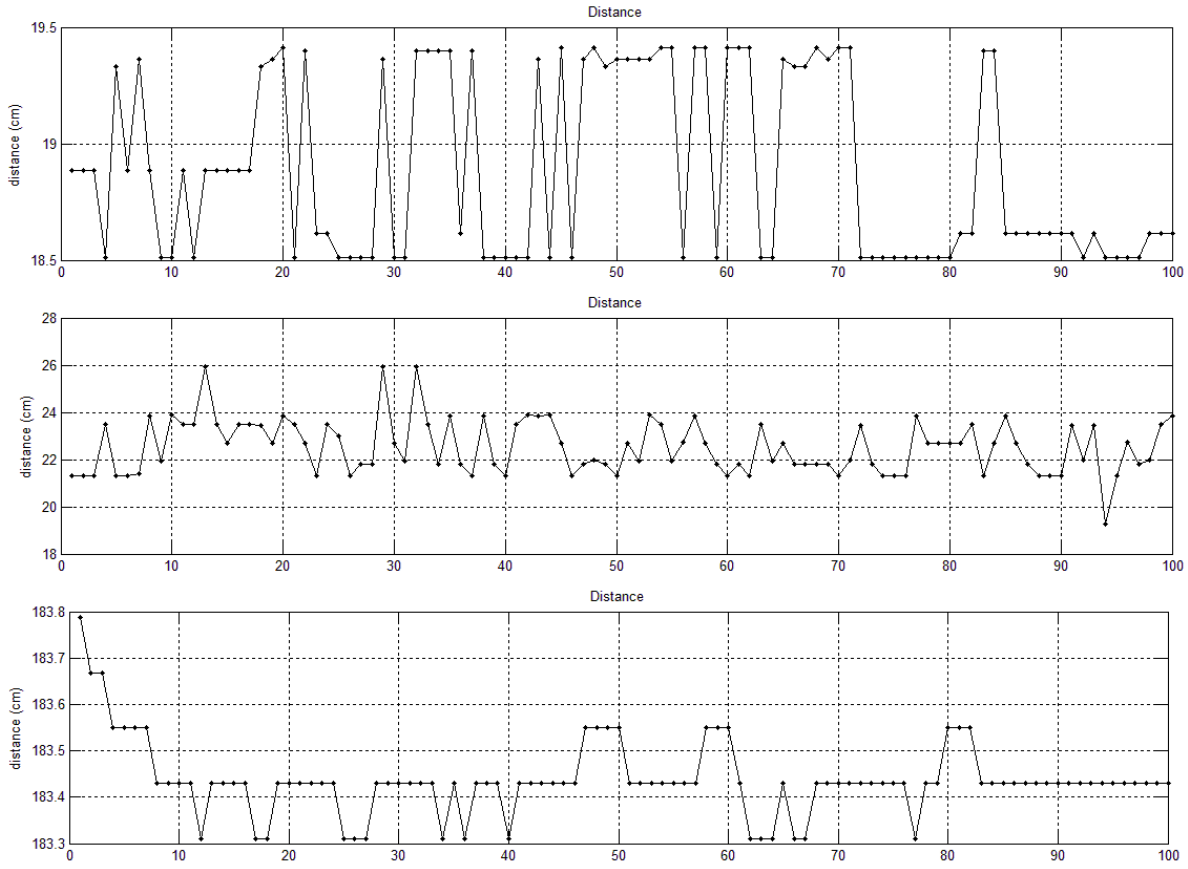



**Figura 3.5** – Distâncias obtidas pelo sonar em três cenários diferentes.

Da análise da **Figura 3.5**, verifica-se que o sensor de distância de ultra-sons tem uma resposta satisfatória para qualquer um dos casos.

No primeiro caso, por se tratar de uma superfície muito favorável para a reflexão do feixe sonoro, obtêm-se medidas bastante estáveis e que se aproximam da distância real.

No segundo caso a distância média também é aceitavelmente próxima da real. Porém neste cenário observa-se uma maior variação das medidas. Isto deve-se ao facto de a superfície reflectora ser de pano, o que a torna irregular e absorve consideravelmente a energia do feixe sonoro.

Finalmente no 3º cenário, salta à vista o facto de as medições serem muito superiores às anteriores, o que faz sentido, visto que a sala onde foi testado o sonar tinha um tecto

alto. Observa-se que mesmo para distâncias superiores, o sensor tem uma resposta muito estável, variando menos de um centímetro.

### 3.3.2. Sensor de Distância de Infravermelhos

Um sensor de distância do tipo infravermelho é constituído por um emissor de luz infravermelha e um detector sensível de posição[5]. Este sensor consegue calcular a distância a um objecto com base no princípio de triangulação **[16]**.

O emissor do dispositivo emite um feixe de luz que "ilumina" um pequeno ponto no obstáculo; no detector, a lente projecta no elemento activo da sua base uma imagem do ponto iluminado. A saída do elemento detector é dependente da posição onde incide a imagem projectada.

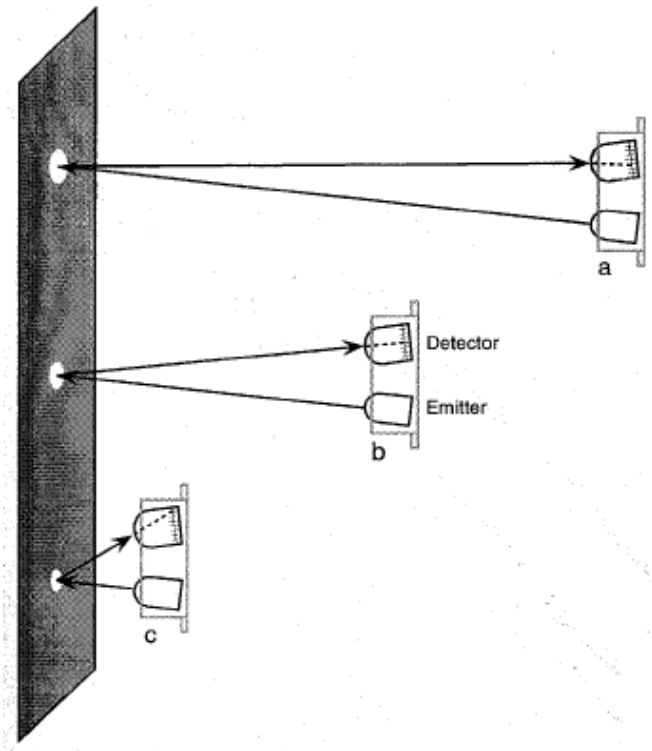



**Figura 3.6** – Esquema de funcionamento do sensor de distância de infravermelhos **[16]**.

Na **Figura 3.6** está ilustrado o funcionamento do sensor de distância IR. Para distâncias superiores, a imagem é projectada mais abaixo no detector, e verifica-se o oposto para distâncias mais próximas. Quando o obstáculo se encontra demasiado perto, o ponto deixa de ser projectado na base do sensor, pelo que não é detectado; para distâncias muito elevadas, além do sensor perder precisão devido ao facto da projecção se deslocar pouco na base do detector, também a energia reflectida torna-se insuficiente para activar o detector, e o objecto não é detectado.

Este sensor é insensível à cor ou textura do objecto em que a luz incide. Porém deve ter-se em conta o facto de que a luz solar interfere directamente com o sensor, pelo que a sua utilização não é adequada no exterior.

[5] Traduzido do inglês "*position sensitive detector*".

**Figura 3.7** – Sensor de Distância SHARP GP2D120 (fonte: http://robosavvy.com/store/product_info.php/products_id/1364).

O sensor usado é o GP2D120 da SHARP, presente na **Figura 3.7**. Segundo o *datasheet*, trata-se de um sensor de medição de distâncias com processamento de sinal integrado e com uma saída analógica em tensão. Este componente possui 3 pinos/fios: alimentação (4.5V a 5.5V), *ground* e saída analógica. Também de acordo com o fabricante, o tempo entre medições consecutivas deve ser de aproximadamente 50ms, de forma a obter valores o mais estáveis possível. Isto reflecte-se numa taxa de amostragem de cerca de 20 amostras por segundo. O intervalo de confiança para as medições é 4cm a 30cm, com uma resposta não linear, como pode ser observado na Figura 3.8. Este sensor é ligado a uma entrada analógica do *Arduino*, cujas ADC's são de 10bits, pelo que é necessário fazer um cálculo para transformar cada valor lido pelo *Arduino* numa distância em centímetros.

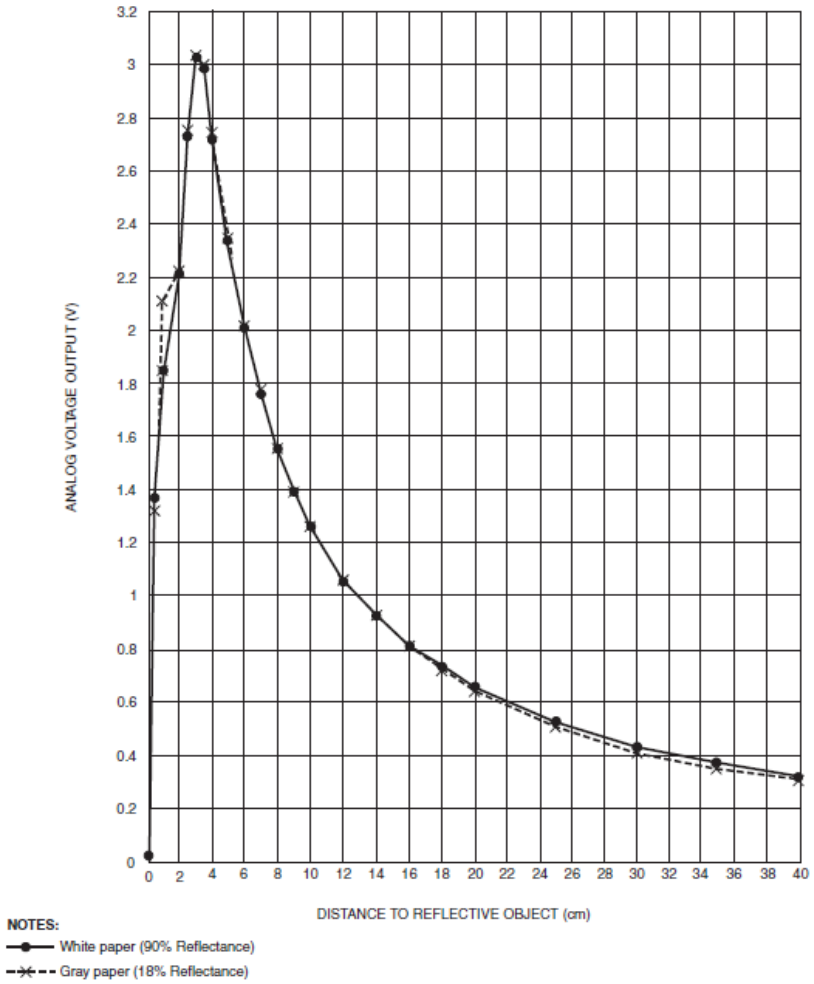



**Figura 3.8** - Curva típica da resposta do sensor GP2D120.

O sensor de distância infravermelho está localizado no mesmo eixo vertical que o sonar, ambos acoplados ao motor servo, assim que ambos sejam sempre direccionados na mesma orientação.

Este sensor tem um alcance mais limitado que o sonar, mas tem uma resposta mais rápida e é mais preciso para curtas distâncias. Ao nível da programação, para minimizar o erro associado à variação das medições, é aplicado um filtro de média. No *Arduino* são adquiridas 5 medidas, é calculada a sua média e feita uma conversão para uma distância. Por fim esta distância (em centímetros) é enviada para o *Matlab* (via *XBee*). Isto implica que a taxa de amostragem deste sensor no *Matlab* é de cerca de 4 amostras por segundo.

Os testes ao sensor incluíram a obtenção de uma fórmula de calibração para converter os valores analógicos lidos pelo *Arduino* para um valor de distância em centímetros. Para tal, foi feita uma aquisição de dados repetitivo, em que o sensor foi colocado a uma distância conhecida de uma superfície conveniente para a reflexão do feixe infravermelho. Para cada distância foi registado o valor analógico correspondente. Por fim traçou-se um gráfico da resposta do sensor, e partindo de uma fórmula inicial encontrada em **[21]**, por tentativa e erro encontrou-se a fórmula que melhor representava o gráfico do sensor infravermelho particular usado no trabalho. Na Figura 3.8, observa-se que para o

intervalo de confiança considerado a curva de resposta do sensor aproxima-se de uma hipérbole, que é do tipo

$$y = \frac{m}{x + a} + b \qquad 3.2$$

Ajustando os parâmetros $m$, $a$ e $b$, a expressão obtida que melhor faz corresponder uma distância em centímetros a cada valor analógico (de 0 a 1023) é a seguinte:

$$distance = \frac{2895}{analog\_value + 12} - 1 \qquad 3.3$$

Na **Figura 3.9** está representado o seu gráfico simultaneamente com o gráfico traçado pelos valores obtidos experimentalmente a cada distância. Note-se que no eixo das ordenadas os valores são de tensão, ao passo que na fórmula e valores experimentais estes são valores inteiros de 0 a 1023. Esta transformação é simples e consiste em redimensionar o intervalo 0-1023 para o intervalo 0-5 de forma directa.

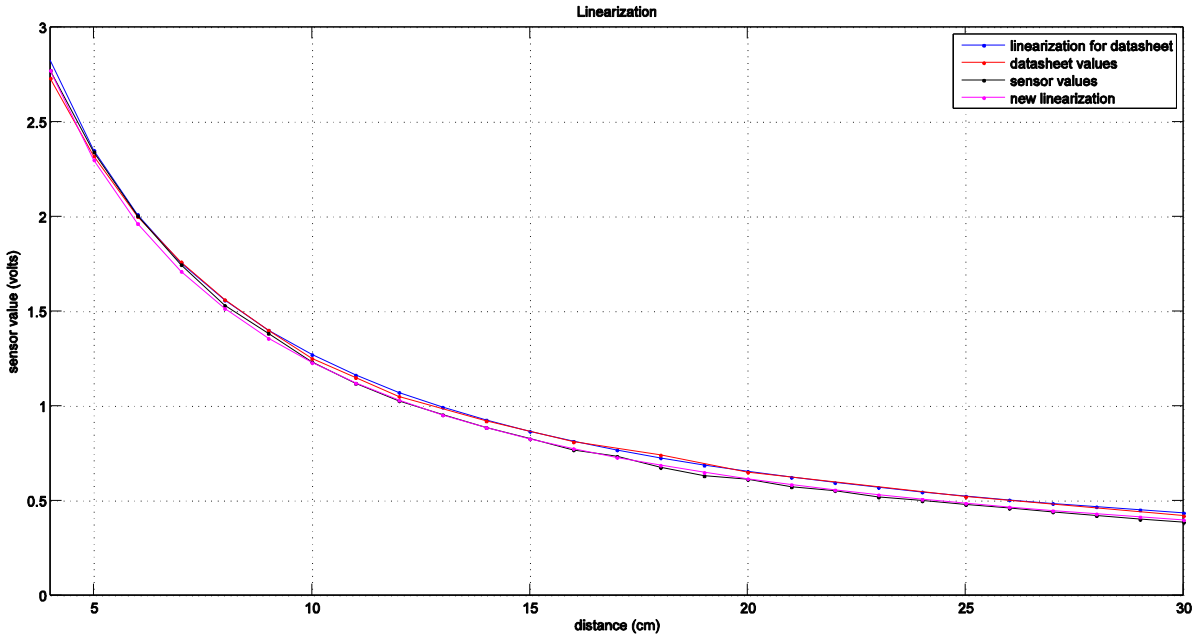



**Figura 3.9** – Gráficos para a conversão de valores analógicos do sensor para distâncias em centímetros.

Verifica-se pelo gráfico acima que ambos os traçados são praticamente coincidentes, o que é completamente satisfatório. É importante mencionar que posteriormente na fase de testes com a navegação, em que o cenário tem menos luz solar (e portanto menos interferência no sensor) do que na presente fase, foram introduzidos dois melhoramentos. Devido às diferenças no ambiente, foi efectuada uma rectificação da fórmula de adequação do sensor de infravermelhos; e as medidas foram convertidas para milímetros, de forma a obter uma melhor resolução do sensor. Estas duas alterações são apresentadas à frente na subsecção **3.7.1**.

Posto isto, os testes são simples, consistindo apenas na fixação do sensor a uma distância de uma superfície reflectora, e observar num gráfico os valores obtidos (ver **Figura 3.10**).

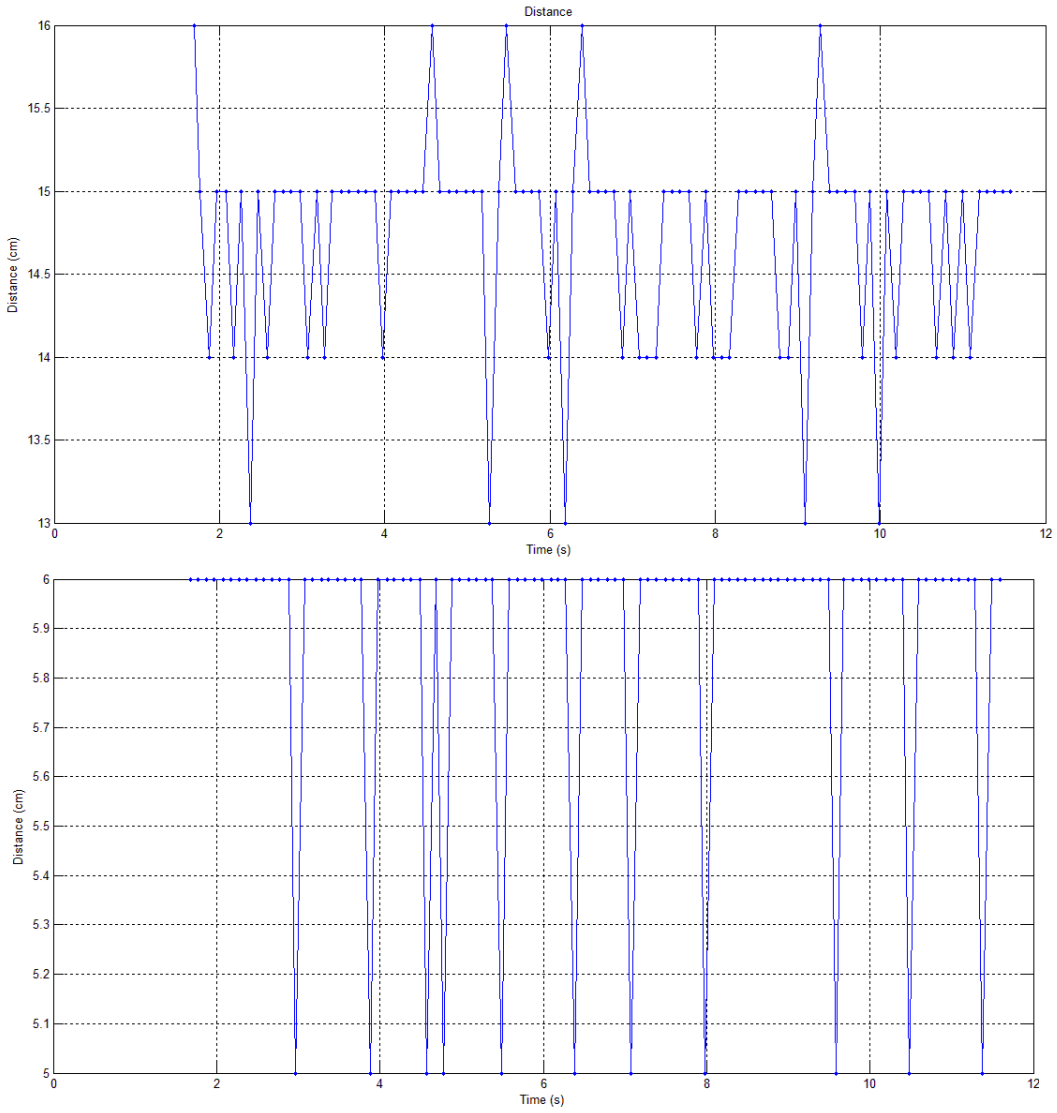



**Figura 3.10** – Distâncias medidas pelo sensor IR quando colocado a cerca de: 15cm do alvo (acima); 6cm do alvo (abaixo).

### 3.3.3. Varrimento com Sensores de Distância

Numa fase já mais avançada do projecto, foi feito um teste que consistiu num "varrimento" frontal de 180° com os dois sensores de distância. Em cada ângulo, foram obtidas várias medidas e aplicado um filtro de média a cada sensor, para minimizar o erro de dispersão. Isto foi conseguido usando o motor servo, que pode ser orientado com precisão ao grau. O veículo robótico foi colocado imóvel em frente a uma parede, e foi feito um varrimento na gama angular de -90° a +90°, em intervalos de 5°. O resultado obtido está na **Figura 3.11**.

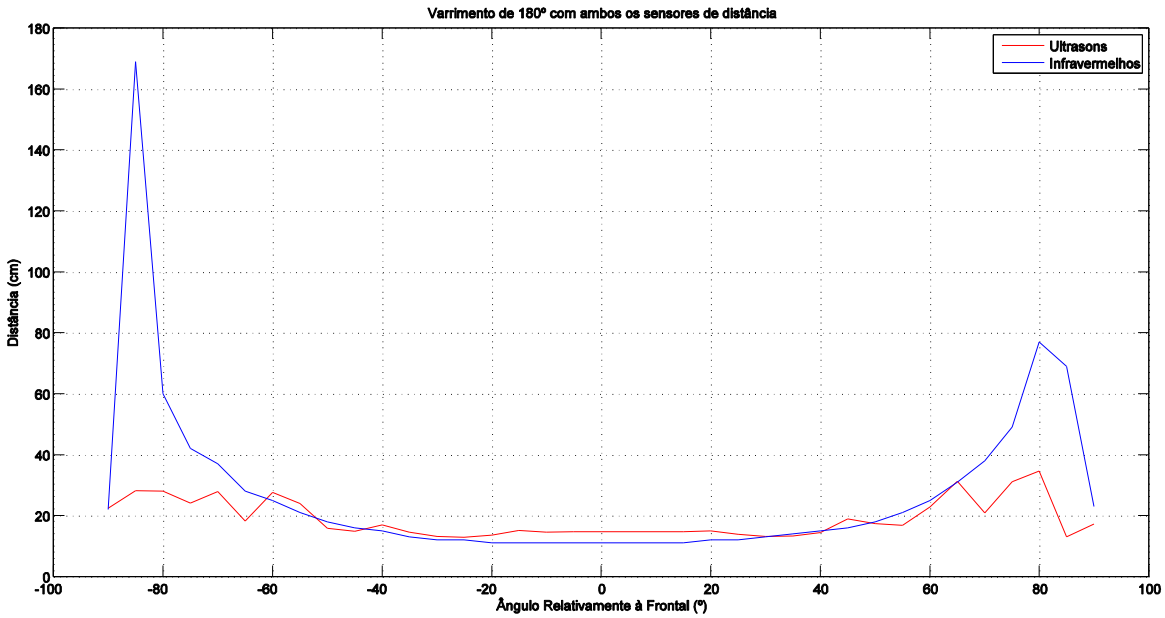



**Figura 3.11** – Medidas obtidas pelos sensores de distância durante o varrimento, a 14cm da parede.

O sensor IR para distâncias curtas, o que se traduz neste caso quando o servo está posicionado entre os ângulos -15° e +15°, é muito estável e tem uma resposta muito próxima da distância real de 14cm. Para ângulos de incidência já bastante afastados da parede frontal, o sensor deixa de detectar a parede e deixa de funcionar correctamente. Já o sonar, no intervalo de -45° a +45° tem uma resposta praticamente constante, devido à forma do feixe emitido pelo sensor, que tem lobos laterais bastante consideráveis, mas ainda assim tendo um mínimo muito próximo dos 0°. Para ângulos mais afastados do centro, verifica-se um aumento da distância, mas não muito considerável, pois como o sensor está relativamente próximo da parede, os lobos laterais detectam-na sempre.

Nos ângulos de incidência perpendiculares à parede os dois sensores estão satisfatoriamente em concordância e próximos da distância real.

### 3.3.4. Bússola Digital

A bússola é um sensor que fornece informação absoluta sobre a orientação do componente no instante em que é lido. Este tipo de sensores é bastante útil para espaços abertos. No entanto, para ambientes *indoor*, levantam-se certos problemas, pois todos os campos magnéticos existentes em volta introduzem erro no sensor, tais como cabos eléctricos, estruturas metálicas ou até mesmo os componentes metálicos do veículo robótico, nomeadamente os motores DC. Existem no mercado alguns sensores preparados para este tipo de problemas, tendo soluções internas para compensar o efeito de campos magnéticos externos **[16]**.

Em concreto, a bússola digital utilizada é a HMC6352, da Honeywell. Segundo o fabricante, trata-se de uma bússola que combina sensores magneto-resistivos de 2 eixos com circuitos analógicos e digitais de suporte necessários. A bússola digital pode ser alimentada com 2.7V a 5.2V, e possui 4 pinos: alimentação, *ground* e os pinos SDA e SCL.

A comunicação com este sensor é feita através do barramento I2C, tal como acontece com o sensor de distância de ultra-sons. O endereço atribuído a este dispositivo

por definição é 0x42, que não provoca conflito com o endereço do sonar, pelo que neste trabalho foram utilizados ambos os endereços de fábrica para cada um destes componentes.

A bússola digital fornece um valor para a orientação segundo o plano horizontal em décimos de grau, isto é, um valor inteiro entre 0 e 3599. O valor 0 corresponde ao norte magnético. De acordo com o *datasheet* do sensor, o tempo mínimo aconselhado entre o pedido de uma nova leitura de orientação ao sensor e a leitura da medida deve ser de 6ms. Isto reflecte-se numa taxa de amostragem máxima de cerca de 167 medidas por segundo, o que é mais do que suficiente, pois este componente está fixo no veículo, e este não efectua rotações tão rápidas. No trabalho, por medidas de precaução, o tempo de espera usado é 7ms.

O sensor foi colocado no veículo robótico o mais horizontalmente possível, pois este é bastante influenciado por inclinações do componente, num plano mais elevado para o manter afastado dos motores DC. Isto porque estes geram campos magnéticos que influenciam as medições da bússola. A localização da bússola pode ser observada na **Figura 3.12.**

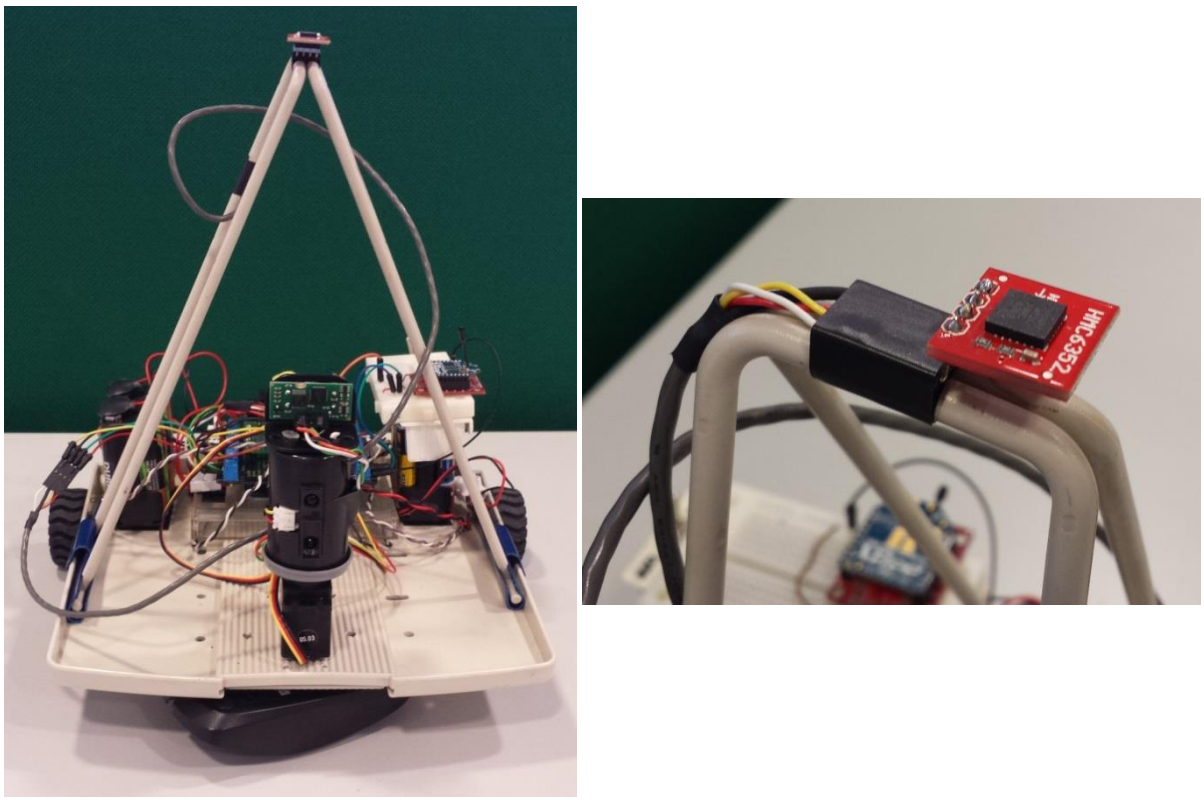


**Figura 3.12** – Localização da bússola. Na imagem da esquerda, está no plano elevado pelas estruturas cinzentas. Na imagem da direita, vê-se a montagem mais em detalhe.

Na programação deste sensor, por forma a aproveitar a sua elevada taxa de amostragem, são adquiridas 10 medições de orientação, calculada a sua média e enviada para o PC através do módulo *XBee*. Este sensor apresenta uma precisão reduzida quando amostrado a elevada frequência (variando alguns graus), mas permite obter medidas estáveis e exactas quando devidamente filtradas. Com o filtro de média, o problema é atenuado ou

praticamente inexistente. A taxa de amostragem recebida no *Matlab* é portanto de cerca de 14,3 medidas por segundo.

Os testes efectuados a este sensor consistiram na fixação do sensor numa dada orientação e observar no gráfico traçado no *Matlab* as medidas obtidas.

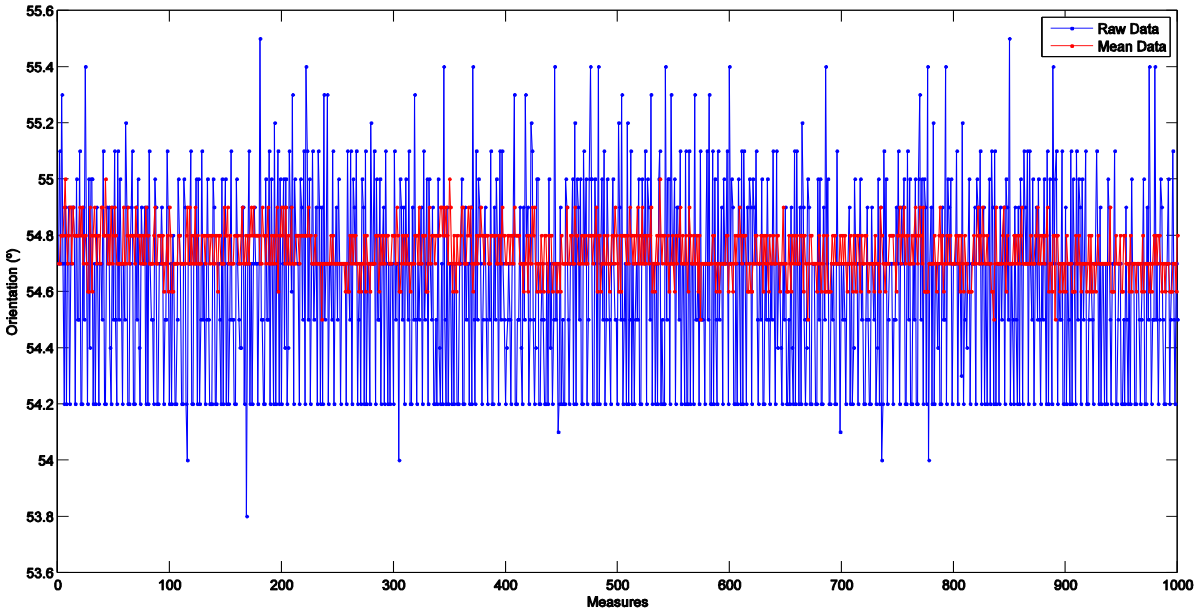



**Figura 3.13** – Medidas de orientação obtidas pela bússola, quando fixa numa posição.

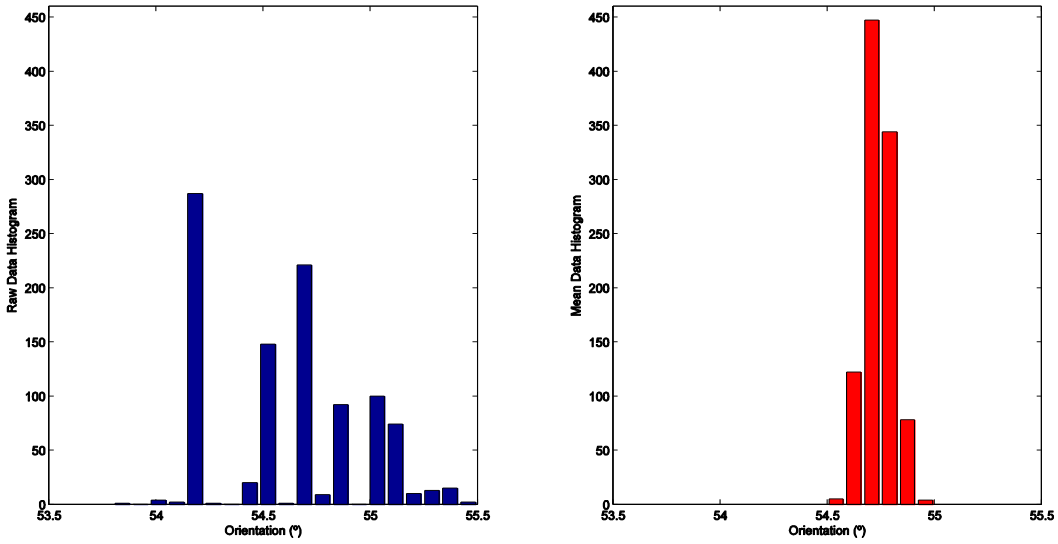



**Figura 3.14** – Histograma das medidas obtidas pela bússola para uma orientação média de 54,7°.

As Figura 3.13 e a Figura 3.14 representam os dados recebidos no *Matlab*: medições de orientação efectuadas pela bússola ligada ao *Arduino*. Em ambos os casos foram recebidas 1000 amostras. A azul estão representados os dados brutos provenientes da bússola, isto é, todas as medidas fornecidas pela bússola ao *Arduino* são enviadas para o *Matlab* e apresentadas no gráfico. A vermelho estão os dados filtrados no *Arduino*, ou seja, este lê 10 medidas de orientação consecutivas, calcula a média entre elas, e envia-a para o *Matlab*.

Na **Figura 3.13** é bem visível a variação dos dados recebidos com e sem filtragem. As medidas a vermelho estão consideravelmente mais compactas, variando apenas cerca de 2 décimos de grau, ao passo que as medidas a azul oscilam praticamente 1 grau.

A **Figura 3.14** ilustra os histogramas das medidas em causa. Esta figura ajuda a interpretar a influência do filtro de média: os dados passam a estar menos espalhados, concentrando-se mais em volta do valor médio da orientação.

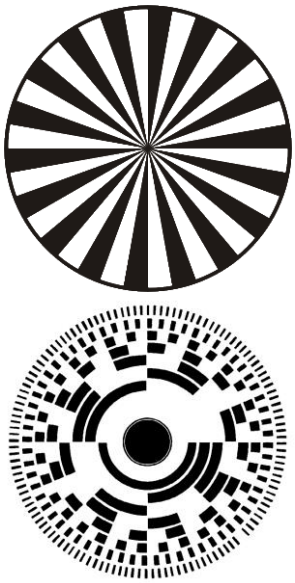
Note-se que apesar de as medidas não estarem indexadas ao tempo, a aquisição com pré filtragem no *Arduino* decorre num intervalo de tempo dez vezes superior ao da aquisição em bruto, pois são obtidas dez vezes mais medidas.

### 3.3.5. *Encoders* acoplados às rodas

*Encoders* são dispositivos electromecânicos que têm a capacidade de contar ou produzir pulsos eléctricos a partir do movimento rotacional do seu eixo. Os *encoders* podem ser absolutos, caso o sensor produza um sinal que corresponde a uma posição do eixo em particular, ou incrementais, se apenas produzem um pulso quando há rotação do eixo **[16]**. Neste caso, os *encoders* usados são incrementais e produzem duas ondas quadradas desfasadas de cerca de 90°, sendo possível identificar o sentido de rotação e ter quatro estados diferentes possíveis em cada período de uma onda **[22]**.

Os *encoders* mais comuns são do tipo fotointerruptor[6], e podem ser observados na **Figura 3.15 a)**. Porém, existem *encoders* designados por fotoreflectores[7], que são os utilizados no presente trabalho. Estes *encoders* em quadratura têm um led infravermelho que emite luz direccionada ao eixo da roda dentada, e que é reflectida para um fototransístor **[16]**. Os *encoders* utilizados neste projecto são incrementais, e são optimizados para as rodas da Pololu, que são as utilizadas. O conjunto da roda e o respectivo *encoder* acoplado está ilustrado na **Figura 3.15 b)**. Cada roda tem 12 "dentes" no aro, o que significa que com as duas ondas quadradas reproduzidas, é possível ter até 48 contagens por revolução (da roda). As ondas produzidas por um *encoder* estão ilustradas na **Figura 3.16**, onde se identificam claramente os 4 estados, bem como a distinção do sentido da rotação graças ao desfasamento.

[6] Traduzido do inglês "*photointerrupter*".
[7] Traduzido do inglês "*photoreflector*".



a)

b)

**Figura 3.15** – Exemplos de *encoders*: **a)** Fotointerruptores incrementais (acima; fonte: http://thedenneys.org/pub/robot/encoders/) e absolutos (abaixo; fonte: http://www.parkermotion.com/dmxreadyv2/blogmanager/blogmanager.asp?post=motor-feedback-options); **b)** Fotoreflectores (*encoder* já acoplado à roda da Pololu; fonte: http://www.robotgear.com.au/Product.aspx/Details/307-Pololu-42-x-19mm-Wheel-and-Encoder-Set).

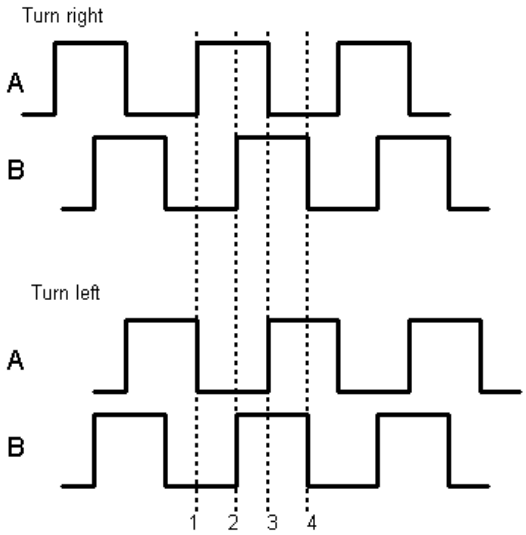



**Figura 3.16** – Sinais gerados pelos *encoders*. Com o desfasamento, sabe-se o sentido de rotação (fonte: http://www.qsl.net/pa3ckr/bascom%20and%20avr/encoders/).

Neste trabalho, os *encoders* são utilizados para contar as rotações de cada roda e assim poder controlar a trajectória do veículo através de um controlador devidamente afinado. As rodas são accionadas através do comando dos motores DC, e como estes não são exactamente iguais, a mesma instrução de velocidade de rotação traduz-se numa rotação real desigual entre as rodas, o que leva o veículo a seguir uma trajectória não linear.

Além disso, o facto de os diâmetros das rodas não serem exactamente iguais, também aumenta a desigualdade da rotação das rodas.

Os *encoders* produzem os pulsos, que são contados no *Arduino*, conectando os sinais produzidos às suas entradas digitais. A contagem e detecção do sentido de rotação de uma roda são feitas da seguinte maneira:

1. Tendo os dois sinais disponíveis em dois pinos do *Arduino*, este apenas tem de ler simultaneamente os estados de ambas as ondas, e quando ocorre uma alteração num deles, regista-se o acontecimento, por exemplo através de um contador.
2. Isto implica que o *Arduino* não pode perder nenhuma transição de estado nas ondas, o que é possível através de dois métodos:
   - × O *Arduino* está constantemente a ler estas duas entradas – o que não é de maneira nenhuma o cenário desejado, pois este é suposto estar a executar outras tarefas e controlar outros dispositivos;
   - ✓ Utilizar interrupções externas para "acordar" o *Arduino* em cada transição de estado, que é de facto a solução implementada. Esta solução funciona mas para conseguir detectar os quatro estados existentes em cada passagem por um "dente" da roda, os dois sinais provenientes de cada *encoder* deveriam estar conectados a entradas de interrupções externas do *Arduino*, o que não possível, pois este possui apenas duas entradas deste tipo.
3. Perante a situação descrita acima, a solução adoptada consiste na utilização de apenas uma entrada de interrupção externa por cada *encoder*. Isto tem como consequência que apenas se detectam as transições de estado de um dos sinais do *encoder*, obtendo-se apenas dois estados em vez dos quatro anteriores, e consequentemente 24 contagens por revolução. Dado que cada roda tem um diâmetro de 4.2mm, tem-se uma resolução de cerca de 5.5mm no deslocamento da roda. A distinção do sentido de rotação das rodas continua a ser possível.

Os aspectos a considerar daqui são:

- As frequências das ondas geradas pelos *encoders* dependem da velocidade de rotação das rodas;
- As fases das ondas variam consoante o sentido de rotação das rodas.

No entanto, o importante nas aplicações que usam *encoders* é detectar com sucesso todas as transições das ondas quadradas, pois são essas que reflectem a rotação da roda no deslocamento.

Na secção **5.4** é apresentado o Modelo de Movimento do *robot* com base na Odometria, e como este usa a informação proveniente dos *encoders* para controlo dos motores segundo uma trajectória.

## 3.4. Processamento de Sinal e Fusão Sensorial

Uma componente muito importante a desbravar no futuro é a fusão sensorial. Esta técnica é muito útil e aumenta bastante a coerência entre os dados sensoriais.

No presente trabalho, o interesse seria fundir os dados provenientes de ambos os sensores de distância, desde que estes estivessem orientados na mesma direcção. Além deste, seria também interessante juntar os valores lidos pelos *encoders* às orientações lidas pela bússola digital.

Com a fusão sensorial, consegue-se atenuar consideravelmente os *outliers* existentes nos dados sensoriais, como o caso da bússola, que é um sensor que apresenta dados muito precisos e estáveis a longo prazo, mas a taxas de aquisição elevadas gera muitos "picos".

O algoritmo de navegação, apresentado na subsecção **4.3**, utiliza fusão sensorial.

## 3.5. Actuadores

### 3.5.1. Motores DC acoplados às Rodas

De um modo geral, um motor eléctrico converte energia eléctrica em energia mecânica. Estes dividem-se em motores AC (corrente alternada) e DC (corrente contínua). Tipicamente, em robótica móvel utilizam-se motores DC, pois a fonte de energia utilizada é normalmente uma bateria DC **[16]**.

Motores DC são motores que recebem um valor de tensão DC variável e geram uma rotação do seu eixo dependente da tensão aplicada e sua polaridade: a amplitude influencia a velocidade de rotação; a polaridade determina o sentido de rotação. No entanto, como são elementos integradores, por razões práticas são muitas vezes controlados por PWM (*Pulse Width Modulation*). Deste modo, recebem e integram ao longo do tempo o valor médio de uma onda quadrada de tensão contínua. Sendo assim, a velocidade de rotação do motor deixa de ser influenciada directamente pelo valor da tensão, mas depende directamente do *duty cycle* da onda quadrada que lhe é aplicada, e é maior quanto maior for o *duty cycle*.

Neste projecto, para além dos motores DC, é também utilizado um *shield* próprio para *Arduino* desenvolvido pela Adafruit para comando de motores. Este *shield* possui entradas para motores DC, motores de passo (*step motors*), e ainda para motores servo. É usada ainda uma biblioteca para *Arduino* desenvolvida pelo mesmo fabricante para a interacção com o *shield*, o que facilita quer a ligação, quer a programação dos motores. Com esta biblioteca podemos abstrair-nos da programação das ondas PWM a enviar aos motores, e indicar simplesmente o sentido e velocidade de rotação no caso dos motores DC, ou qual o ângulo em que pretendemos orientar o motor servo.

Cada roda tem acoplado um motor DC, e cada um destes pode ser rodado nos dois sentidos, de forma independente. Daqui resulta o modo de locomoção diferencial, isto é, a trajectória que o veículo robótico toma depende da velocidade e sentido de cada uma das rodas. Este facto tem vantagens e desvantagens. Por um lado, devido aos motores não serem iguais e gerarem saídas diferentes para entradas iguais, o veículo toma uma trajectória que pode não ser a desejada. Este problema é resolvido com a aplicação de controlo dos motores com realimentação, como é apresentado na subsecção **3.6**. Por outro lado, esta independência dos motores (e consequentemente das rodas) possibilita que o veículo rode em torno de si próprio, o que é muito conveniente em veículos robóticos deste tipo. Estes veículos são denominados não-holonómicos, pois não se podem deslocar (instantaneamente)

em qualquer direcção. Por outro lado, os sistemas de locomoção *synchro-drive* e omnidireccional são exemplos de *robots* holonómicos **[23]**.

A acção dos motores é apresentada na subsecção **3.6**, em que a velocidade aplicada a cada roda (e motor) é actualizada constantemente com base nos desvios da trajectória.

### 3.5.2. Motor Servo

Um servomotor é um actuador que permite o controlo preciso de posição angular. Este tipo de motor inclui um motor DC, engrenagens, limitadores à rotação máxima do eixo, um potenciómetro no eixo para *feedback* da posição angular, e um circuito integrado para controlo da posição angular **[16]**.


**Figura 3.17** – Servomotor utilizado no projecto, HS-322HD da HiTEC (fonte: http://www.abra-electronics.com/products/Hitec-HS%252d322HD-Standard-Deluxe-Servo-.html).

O servo utilizado no projecto é o HS-322HD da HiTEC, apresentado na **Figura 3.17**. Este motor é controlado através de pulsos PWM, e tem três fios: alimentação, *ground* e sinal de controlo. Tal como os motores DC, também este motor é ligado ao *shield* da Adafruit. Porém, neste caso a ligação trata-se apenas de um *jumper*, e o servo é comandado e alimentado directamente pelo *Arduino*. Usando a biblioteca nativa do *Arduino* para Servos, o comando deste motor resume-se à execução uma linha de código na qual se especifica a posição angular em que se pretende colocar o servomotor.

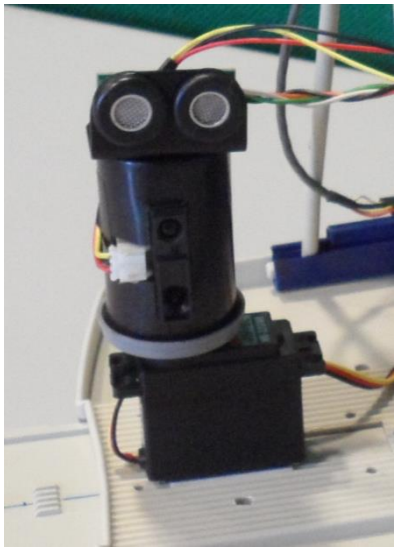


**Figura 3.18** – Fixação dos sensores de distância ao servomotor.

Tal como se pode ver na **Figura 3.18**, os sensores de distância estão encapsulados numa caixa de rolo de fotográfico que está afixada ao eixo do motor. Na figura, os sensores estão orientados na mesma orientação angular, e foi nesta configuração que foram obtidos os dados para o varrimento apresentados na secção **3.3.3**.

## 3.6. Controlo de Motores (PID)

Quando se pretende controlar, ou compensar um sistema, fala-se em controlo, ou compensação do sistema. A unidade que efectua essa acção tem o nome de controlador, ou compensador, e pode ser implementada em *software* ou *hardware*.

O controlo pode ser aplicado e encontrado em diversas áreas e cenários: um sistema de ar condicionado é um controlador de temperatura; quando conduzimos, estamos na realidade a executar um controlo de trajectória, etc. De um modo simplista, fazer o controlo consiste em comparar o estado actual do sistema que se pretende controlar com o estado que se pretende que ele tenha, e actua-se por forma a contrariar a diferença que possa existir.

No caso específico deste trabalho, o controlo efectuado é sobre a trajectória do veículo. Devido às diferenças existentes entre os motores acoplados às rodas, o veículo adquire uma trajectória errática. Utilizando os *encoders* já referidos anteriormente, consegue-se saber qual o deslocamento de cada roda, e com base na diferença entre elas, aplicam-se novas velocidades às rodas por forma a corrigir a trajectória. Com este método, é possível manter o veículo robótico a seguir qualquer trajectória.

Existem muitos tipos de compensadores, com base na teoria clássica ou moderna, com amostragem contínua ou discreta. Todos têm as suas vantagens e desvantagens. No caso deste projecto, o controlador implementado é do tipo PID, devido à sua simplicidade de codificação e bom desempenho neste tipo de aplicações.

Um controlador do tipo PID consiste em três componentes – Proporcional, Integral e Derivativo – e o seu diagrama de blocos típico num sistema está representado na **Figura 3.19**. Este tipo de compensadores tende a anular o erro existente entre a saída e entrada

(referência) do sistema, actuando em cada uma das suas componentes (proporcional, integral e derivativa), colocando "pesos" em cada uma através de constantes associadas, denominadas Kp, Ki e Kd, respectivamente.

Estes controladores são largamente utilizados em ambientes industriais, tal como controlo de temperatura.

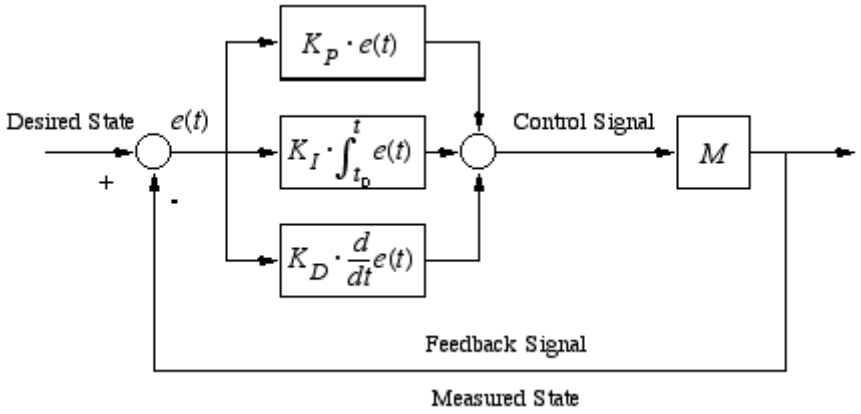



**Figura 3.19** – Diagrama de blocos típico de um sistema com um controlador PID (fonte: http://cs.brown.edu/~tld/courses/cs148/02/sensors.html).

A componente proporcional do controlador pretende anular o erro actual do sistema. A componente integral tem como objectivo contrariar o erro acumulado do sistema; é mais lenta a actuar e geralmente é activada apenas quando a saída se começa a afastar constantemente da referência. A componente derivativa pretende antecipar as variações do erro, actuando no sentido de as contrariar; é a componente mais crítica, pois se a saída for muito oscilante, ainda que sejam pequenos desvios, vai causar grande instabilidade no sistema.

A amostragem no tempo do controlador PID neste trabalho é discreta. Isto significa que o erro do sistema é calculado em intervalos de tempo constantes e também são actualizadas as velocidades a aplicar a cada roda. Esta frequência de amostragem também tem consequências na resposta do sistema: caso seja baixa, o sistema demora a actualizar a velocidade e acaba por não corrigir correctamente e/ou atempadamente a trajectória do veículo; uma sobre-amostragem amplifica as pequenas oscilações do sistema e tende a corrigir desvios que na realidade são "inofensivos" em termos de longa distância, levando à instabilidade do sistema.

O controlador PID foi implementado em *software*. De um modo mais detalhado, o controlo da trajectória rectilínea do veículo robótico funciona da seguinte forma:

- Após alguns testes, apurou-se que uma boa frequência para o cálculo do erro e posterior actualização da velocidade das rodas é de 10Hz, isto é, a velocidade de cada roda é actualizada a cada 100ms.
- Com a ajuda dos *encoders*, é mantida numa variável (para cada roda) a contagem de impulsos actual de cada roda. Lembre-se que cada contagem corresponde a um deslocamento de 5,5mm.
- A cada 100ms, os valores destas variáveis contadoras são comparados, é calculada e armazenada a diferença entre elas (sendo este o erro que se quer

compensar), e os contadores são recolocados a zero e aptos e recomeçar a contagem de impulsos imediatamente.

- Este erro é então processado pelo PID, calculando as suas componentes, e é traduzido em velocidades a aplicar a cada uma das rodas.

Existem alguns métodos para corrigir a trajectória de um veículo de *drive* diferencial actuando independentemente nas velocidades das duas rodas. Nesta implementação, a trajectória é corrigida diminuindo (ou até mesmo anulando por completo) a velocidade de uma das rodas (consoante o lado para o qual o veículo deve curvar) e mantendo ao máximo (de referência) a velocidade da outra roda.

O controlo numa trajectória rectilínea aborda três aspectos importantes: a contagem dos impulsos por parte dos *encoders*; o cálculo do erro na trajectória, ou seja, o controlo com o PID; e o comando dos motores DC acoplados às rodas com as velocidades calculadas pelo PID.

No capítulo **5** é estudado com mais detalhe o controlo dos motores. Na subsecção **5.5.1** efectua-se o controlo sobre uma trajectória rectilínea, e na subsecção **5.5.2** estuda-se o controlo no seguimento de uma parede, em que se corrige a distância do *robot* à parede. Na secção **5.6** é analisado o controlo num percurso mais longo, que engloba as duas situações anteriores.

## 3.7. Melhoramentos nos Sensores de Distância

### 3.7.1. Infravermelhos

Tal como foi referido na subsecção **3.3.2** acerca do sensor de infravermelhos, após ter a expressão para conversão dos valores analógicos para medidas de distâncias em centímetros, foram feitos alguns ajustes, nomeadamente para converter as medidas para milímetros, e para corrigir algum erro proporcional ainda existente.

A correcção do erro foi feita de modo grosseiro, relacionando o erro (entre a distância real e a distância obtida pelo sensor) com o aumento da distância ao obstáculo. Posto isto, a nova expressão para a conversão das grandezas é dividida em duas partes, e é obtida pelas seguintes expressões.

$$distance = \frac{28950}{analog\_value + 12} - 10 \qquad \textbf{3.4}$$

$$distance = distance + distance/10 \qquad \textbf{3.5}$$

Com esta nova conversão foram adquiridas de forma sistemática várias medidas de distância, sendo registadas também as distâncias reais. Assim foi possível calcular com alguma precisão o valor médio e desvio padrão do erro do sensor com esta melhoria.

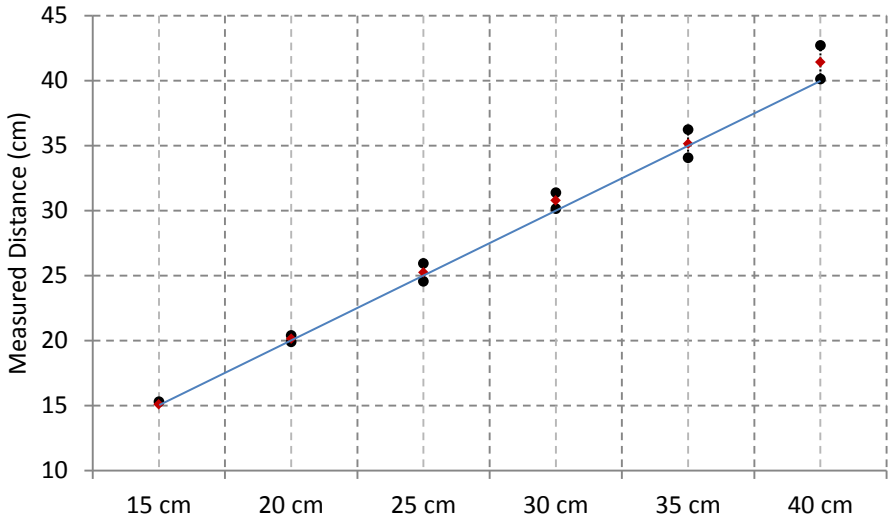



**Figura 3.20** – Valor médio e variação com base no desvio padrão das medições em função da distância a um obstáculo.

Conforme pode ser observado na **Figura 3.20**, o sensor apresenta um ligeiro *bias* que aumenta com a distância. As medições tornam-se também mais dispersas com o aumento da distância, mas no geral variam pouco.

### 3.7.2. Ultrasons

Conforme foi referido na secção **3.3.1**, um sonar *range finder* tem tipicamente um feixe de acordo com a **Figura 3.1 a)**, que tem uma abertura angular considerável e lobos laterais que podem ter impacto nas medições. Na aplicação presente isto traduzia-se na reflexão do feixe acústico no pavimento sempre que distância do sensor à parede fosse superior a aproximadamente um metro, introduzindo assim erros grosseiros nas medições de distância. Este problema pode ser mitigado com a aplicação de um material absorvente acústico em torno dos transdutores do sonar. Neste caso, foi aplicado um material felpudo conforme pode ser observado na **Figura 3.21**. Este material isolador não pode ser muito comprido, para reduzir apenas os lobos laterais, não interferindo significativamente com o feixe frontal.

**Figura 3.21** – Sonar após aplicação de material isolador acústico em torno dos transdutores.

Com esta solução, o sonar *range finder* passou a ter uma resolução angular bastante superior, como se pretendia. Analogamente ao que foi feito com o sensor de infravermelhos, foram recolhidas medidas de distância a uma parede, para serem comparadas com as distâncias reais. Com estes dados, caracterizou-se o erro do sensor na **Figura 3.22**.

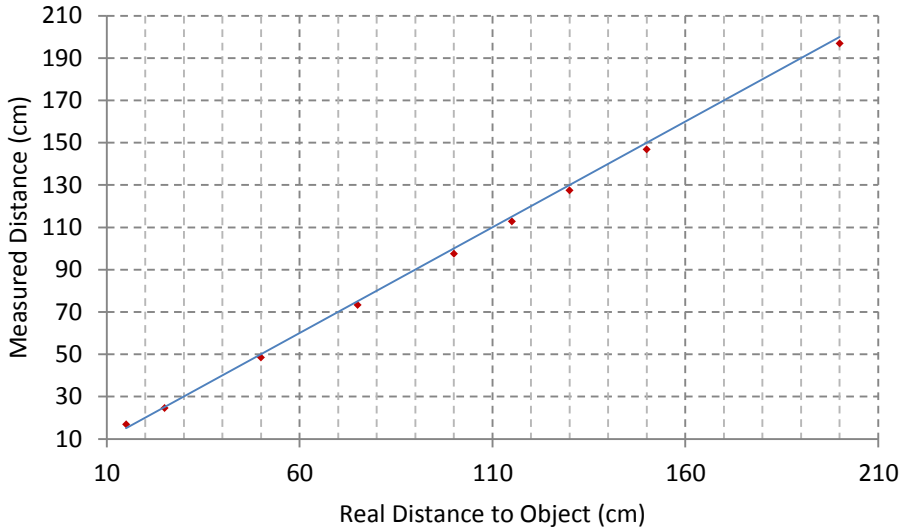



**Figura 3.22** – Valor médio das medições em função da distância a um obstáculo. A variação das mediadas, apesar de estar representada, é demasiado pequena para ser visível (desvio padrão tipicamente da ordem do milímetro).

Finalmente, verifica-se que o erro do sensor de distância de ultra-sons afasta-se mais da distância real na menor distância (15cm). O valor médio das medições, aparentemente, não depende da distância e não é muito significativo. Apresenta também um desvio padrão muito reduzido, inferior a 3mm, sendo inferior a um milímetro na maioria dos casos.

# 4. *Software* de Navegação e Controlo

## 4.1. Introdução

Em robótica, navegação pode ser definida como a combinação das três competências fundamentais:

- Localização
- Planeamento de um trajecto
- Construção de um mapa e sua interpretação

Localização denota a habilidade de o *robot* estabelecer a sua própria pose (posição e orientação) no mapa de referência. Planeamento de um trajecto representa uma extensão à localização, dado que requer a determinação da posição actual do *robot* e a posição destino desejada, ambas em relação ao mapa de referência. O planeamento engloba ainda uma eventual escolha de trajecto, como o percurso mais curto, ou mais rápido. Construção do mapa é a concretização do modelo do ambiente que se tem, e respectiva representação dos seus elementos em termos de um mapa métrico ou outro modo de representação.

De um modo geral, existem dois tipos de navegação:

- Navegação *Indoor*
- Navegação *Outdoor*

No caso do presente trabalho, trata-se de navegação *indoor*, pois a navegação é feita num espaço confinado dentro de um edifício.

Muitas vezes, a navegação está directamente relacionada com o mapeamento, que é o processo de construir o mapa do ambiente. Uma técnica de navegação avançada, cada vez mais utilizada em robótica móvel é o SLAM. Esta técnica é usada por *robots* e veículos autónomos para construir um mapa de um dado ambiente (desconhecido à priori), ou para actualizar um mapa dentro de um ambiente conhecido, ao mesmo tempo que é determinada a sua posição no mapa.

Apesar de o veículo robótico construído neste trabalho ter a característica de ser autónomo, não é operado sempre como tal. Isto é, o veículo robótico executa as suas tarefas e comportamentos de forma autónoma, mas geralmente os comportamentos a tomar em cada instante não são decididos por si. O veículo robótico é comandado através de instruções enviadas a partir de um interface gráfico desenvolvido no PC. Neste interface é possível enviar os comandos com os respectivos parâmetros do comportamento e visualizar em tempo real os dados sensoriais que vão sendo adquiridos pelo *robot*.

Os dados adquiridos pelo *robot* durante a execução de um percurso são guardados no PC e processados no algoritmo de navegação. Neste algoritmo já está disponível um

mapa do ambiente onde o veículo se desloca, e é representado o trajecto que ele toma ao longo do tempo. A navegação aqui apena visa a componente de localização do veículo robótico e sua representação no mapa, não havendo nenhum planeamento de trajectórias.

De modo a ser bem estruturado, um *robot* deve ter uma certa arquitectura de *software*, que organiza e dita o modo como o agente robótico age conforme o ambiente e o seu conhecimento acerca dele. Algumas definições de arquitectura robótica são:

- Uma forma de princípios de organizar um sistema de controlo. No entanto, além de providenciar a estrutura, impõe também restrições ao modo como o problema de controlo pode ser resolvido **[24]**
- Arquitectura robótica normalmente refere-se a *software*, ao invés de *hardware* **[25]**
- Modo como a tarefa de gerar acções a partir de percepções é organizada **[26]**.

As componentes de arquitecturas robóticas incluem: representação, controlo e coordenação, aprendizagem, desempenho temporal, inspiração biológica e psicológica, e avaliação. Pesando de maneira diferente todas estas vertentes podem ter-se várias abordagens de arquitecturas. Nos extremos podemos ter um controlo puramente deliberativo, em que cada acção a ser tomada pelo *robot* implica uma grande ponderação; ou contrariamente pode ter-se um controlo puramente reactivo, em que o *robot* não pensa sequer, limitando-se simplesmente a reagir. Num plano intermédio considera-se um controlo híbrido, que explora um pouco das duas perspectivas anteriores. Este esquema está bem ilustrado na **Figura 4.1**.

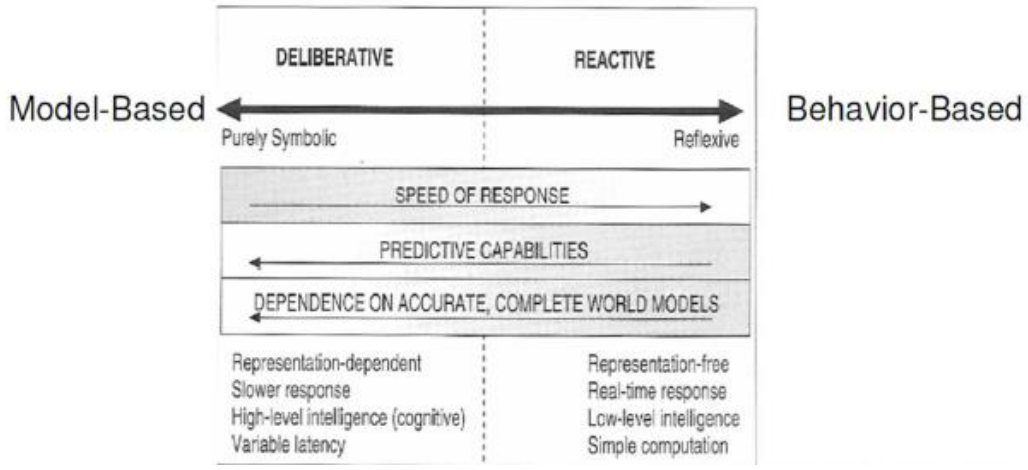



**Figura 4.1** - Arquitectura Deliberativa vs. Arquitectura Reactiva **[25]**.

As organizações típicas de arquitecturas e suas características estão resumidas na **Tabela 4.1.**

**Tabela 4.1** – Características das arquitecturas mais comuns em robótica móvel.

| | |
|---|---|
| **Deliberativa** | • Construir mapas<br>• Seleccionar comportamentos<br>• Monitorizar desempenho<br>• Planeamento<br>• Paradigma deliberativo/reactivo híbrido |
| **Reactiva** | • Baixos requisitos de processamento, e barato<br>• Sem modelo do mundo |
| **Baseada em Comportamentos** | • Combinação de comportamentos simples<br>• Sem modelo centralizado do mundo<br>• Cada comportamento pode armazenar a sua própria representação |
| **Híbrida** | • Combina abordagens reactiva e deliberativa |

Neste trabalho, a arquitectura adoptada (por *software*) para organizar o veículo robótico é a arquitectura baseada em comportamentos, e é apenas esta que é abordada daqui para a frente.

O esquema da arquitectura funcional do veículo robótico, adaptado de **[27]**, está representado na **Figura 4.2**.

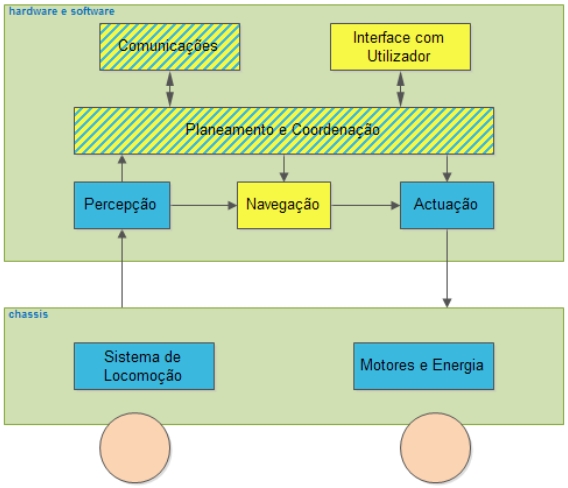



**Figura 4.2** – Arquitectura funcional do veículo robótico.

Esta arquitectura representa o sistema robótico global, isto é, o *robot* (baseado em *Arduino*) e o PC. Os blocos a azul foram implementados no kit robótico, os blocos a amarelo foram implementados no PC. Já os componentes de "Comunicações" e "Planeamento e Coordenação" foram implementados (ainda que minimamente) em ambos.

No *Arduino*, o bloco de "Planeamento e Coordenação", à parte de uma pequena coordenação entre os comportamentos a implementar pelo *robot*, não foi implementado por duas razões:

- Porque o objectivo do trabalho da tese era implementar outros blocos funcionais, incluindo os de "Percepção", "Actuação" e "Comunicações", e seria inviável no curto período de tempo disponível ir além disso;
- Porque as funções de "Planeamento" e "Coordenação", tal como a "Navegação" implicam recursos de *software* que dificilmente seriam

conseguidos numa plataforma baseada num *Arduino*; deixaram-se portanto essas funções para serem executadas num outro sistema que actualmente é constituído por um PC, mas poderia ser um *single-board computer*, como o *Raspberry Pi*.

É ainda de salientar a importância do bloco de "Comunicações" estar presente em todos os robots e também no PC. Não só para o PC poder receber as informações de cada kit robótico e executar o bloco de "Navegação", mas também para usar os vários veículos robóticos como pontos intermediários de comunicação. Por exemplo, os *Mars Rovers* em Marte usam as sondas *Mars Odissey* e *Mars Global Surveyor* como intermediários para comunicar com a Terra (por questões energéticas e também porque as sondas estão mais tempo no campo de visão da Terra do que os *robots*).

## 4.2. Programação por comportamentos

De uma maneira simplista, um comportamento é uma reacção a um estímulo **[25]**. Nesta perspectiva, um comportamento descreve o modo como um *robot* deve interagir com o seu ambiente. Esta ideia aproxima-se de um sistema robótico puramente reactivo, que liga a percepção (sensorial) à acção de uma maneira muito "próxima", sem a intervenção de representações abstractas ou historial temporal **[25]**.

Também na literatura, em **[28]** e em **[25]** os comportamentos de Braitenberg são descritos como a presença de intenção nas acções do *robot*. Sem intenção, os elementos actuadores simplesmente reagem de forma imprevisível aos sinais eléctricos gerados pelos elementos sensoriais.

A programação (ou controlo) por comportamentos advém da arquitectura de subsunção de Brooks (ver **Figura 4.4**), na qual os dados sensoriais não são tratados directamente, mas accionam comportamentos no *robot* (ver por exemplo **[16]**) Neste contexto, comportamentos são camadas de controlo com diferentes níveis de prioridade que correm em paralelo. Na realidade, os comportamentos de maior prioridade suprimem os inferiores. Quando os comportamentos de maior prioridade deixam de estar activos por alguma condição (sensorial), voltam a activar os comportamentos de níveis inferiores. Os comportamentos podem ainda ser interpretados como máquinas de estado finito. Na **Figura 4.3**, o comportamento *Escape* do *robot* "*Rug Warrior*" (desenvolvido em **[16]**), destinado a resolver uma situação de colisão com um obstáculo, é representado por uma máquina de quatro estados.

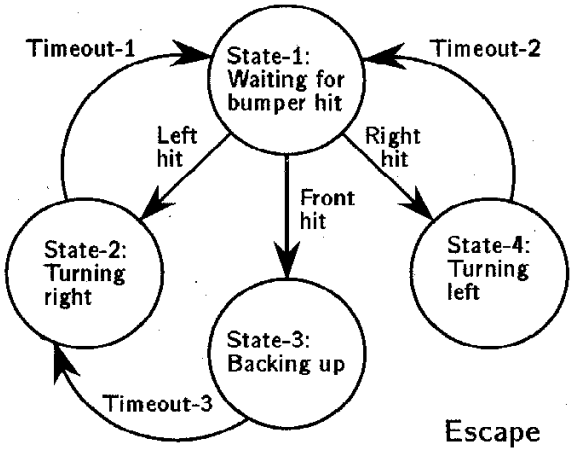



**Figura 4.3** – Máquina de estados do comportamento *Escape* do *robot* "*Rug Warrior*" **[16]**.

Uma arquitectura baseada em comportamentos assenta em princípios como **[16, 25, 28, 29]**:

- Comportamentos são implementados como leis de controlo;
- Cada comportamento recebe *inputs* dos sensores do *robot* e/ou de outros módulos, e gera *outputs* para os actuadores e/ou outros módulos;
- Diferentes comportamentos podem receber dados provenientes dos mesmos sensores e actuar nos mesmos actuadores;
- Comportamentos são codificados de uma maneira simples e são adicionados de forma incremental;
- Comportamentos (ou subconjuntos) são executados de forma concorrente (apesar de neste trabalho este cenário não ser adoptado, pelo que).

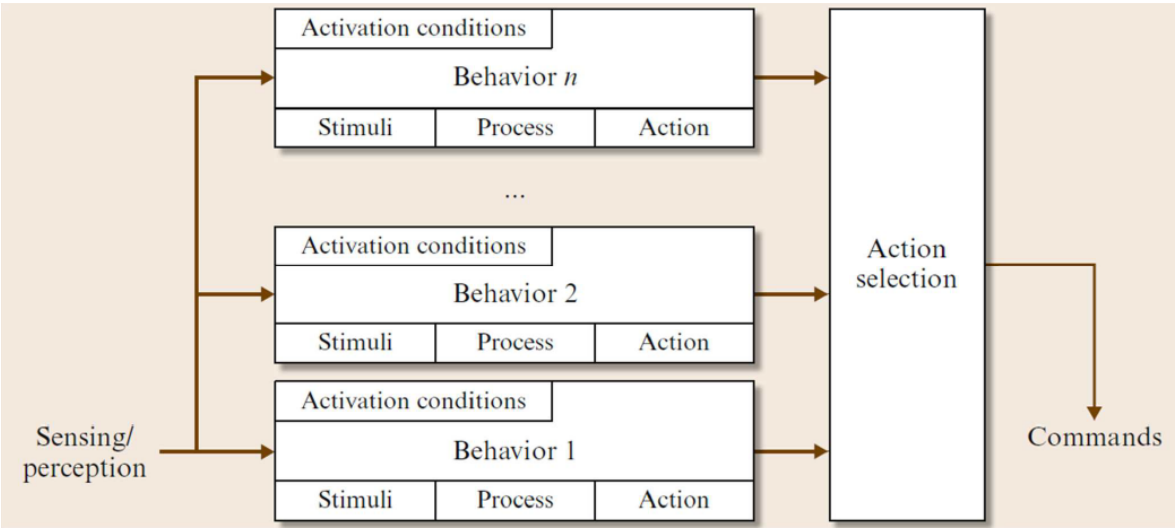



**Figura 4.4** – Esquema de funcionamento de uma arquitectura baseada em comportamentos **[29]**.

### 4.2.1. Comportamentos implementados

Normalmente, as arquitecturas de controlo por comportamentos ditam que estes podem ser executados em paralelo, podendo estar activos mais do que um simultaneamente.

Porém, neste trabalho é executado apenas um comportamento de cada vez, e foram implementados por *software* os seguintes no veículo robótico:

- IDLE;
- STRAIGHT;
- STOP;
- SCANNING;
- FOLLOWALL;
- TURN;
- ANGLE.

Este conjunto de comportamentos permite ao *robot* executar uma série de tarefas. Adverte-se o leitor que dada a natureza básica das acções associadas a cada comportamento, considera-se que cada comportamento corresponde a um estado. Isto corresponde a uma simplificação do conceito de comportamento que normalmente inclui vários estados e é representado por uma Máquina de Estados Finitos conforme adiante é descrito. No ponto actual do trabalho, a transição entre os estados/comportamentos é desencadeada maioritariamente por instruções externas recebidas pelo *robot*. Porém, alguns comportamentos podem passar automaticamente para outros, como o STOP. Este comportamento trata de abrandar e parar os motores das rodas, desencadeando a activação do estado IDLE (estado base/neutro) quando termina a tarefa de paragem.

Nos comportamentos STRAIGHT e FOLLOWALL, além do seguimento das trajectórias, são adquiridos e enviados periodicamente dados sensoriais. Tal como foi já apresentado anteriormente no Capítulo **2.**, cada envio de uma trama *XBee* necessita de um tempo não desprezável – cerca de 20ms a 30ms, dependendo do tamanho das tramas – que introduzem atrasos no processamento no *Arduino*. Para mitigar o efeito destes atrasos, que é bastante crítico, os dados sensoriais adquiridos são organizados em pacotes de dados com várias medidas (descritos na subsecção **4.5.1.1**), por forma a evitar uma elevada quantidade de transmissões de *frames ZigBee* e reduzindo assim os atrasos introduzidos por estes. No *robot*, a cada 100ms é guardada uma medida de cada sensor (a mais recente), e quando são obtidas cinco amostras de cada, é associada uma *tag* temporal ao pacote de dados e este é enviado para o PC via módulo *XBee*. O mesmo acontece com os dados dos *encoders*. Como a taxa de amostragem dos sensores no *Arduino* é muito estável e precisa, basta associar um instante temporal a cada cinco medidas e, no PC, assumir que estas são uniformemente distribuídas em cada intervalo de tempo. Com esta implementação, reduzem-se atrasos associados ao envio massivo de dados, e mantém-se uma taxa de amostragem satisfatória e constante no PC. A alternativa a esta solução seria atribuir um "selo" temporal a cada medida, mas isso tornaria o pacote de dados demasiado grande, o que se traduz no aumento do tempo do seu envio e consequente atraso no processamento no *Arduino*.

A interacção entre comportamentos pode ser interpretada e representada como uma máquina de estados finitos. A **Figura 4.5** ilustra os comportamentos implementados no sistema actual e como estes se relacionam.

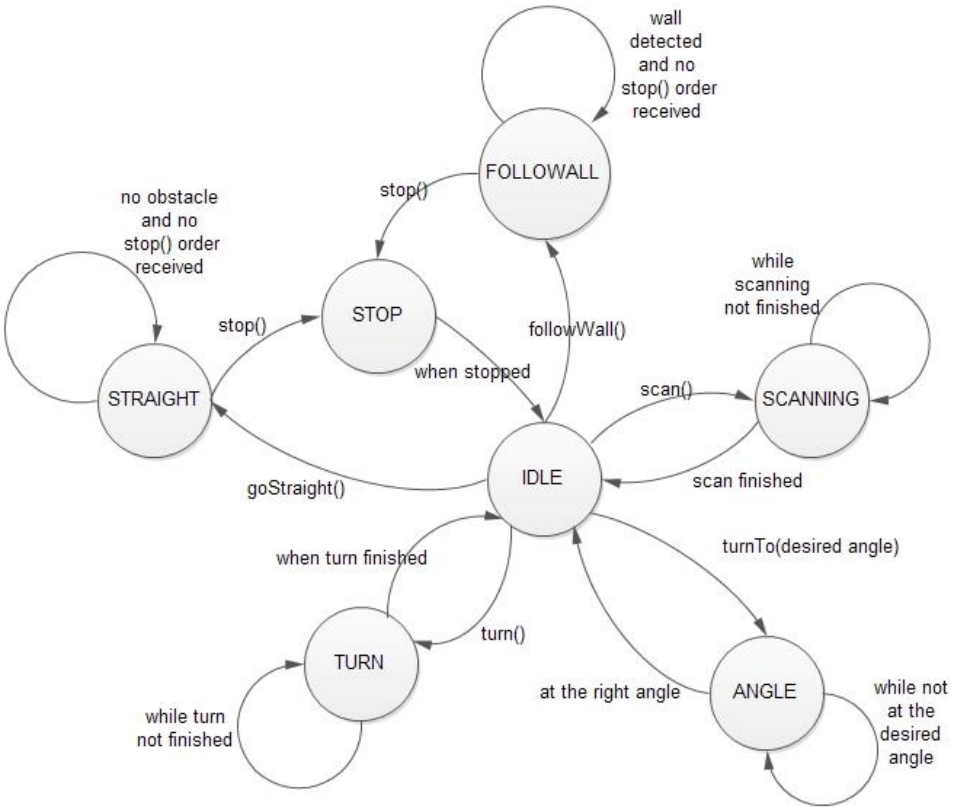



**Figura 4.5** – Diagrama de estados dos comportamentos implementados no *robot*.

Como se pode ver a partir do diagrama de estados apresentado, a partir do estado base IDLE é possível transitar para qualquer um dos outros (excepto para o estado STOP, o que faz sentido, pois o veículo já está imobilizado). Os estados TURN e ANGLE actuam nos motores das rodas de forma a orientar o veículo de forma desejada, transitando automaticamente para o estado IDLE finda a tarefa. Os estados FOLLOWALL e STRAIGHT, dado que implicam movimento contínuo do veículo, passam ao estado STOP quando pretendem transitar para o comportamento neutro IDLE. O estado de SCANNING implica que o veículo esteja parado, daí que apenas se possa transitar para este quando o estado actual é o IDLE; executa a sua tarefa com o *robot* imobilizado e volta ao estado base IDLE quando termina.

Resumindo, todos os estados transitam apenas do estado neutro IDLE (excepto STOP), e quando terminam a tarefa retornam a ele directa ou indirectamente (caso do FOLLOWALL e STRAIGHT).

Passa-se então a enumerar e explicar cada um dos comportamentos implementados na arquitectura.

#### 4.2.1.1. IDLE

Este comportamento corresponde ao estado base do diagrama. Qualquer estado é direccionado a partir daqui, e todos retornam a ele no final das suas tarefas. Este estado implementa um pequeno comportamento de poupança de energia, pois nada faz a não ser

reiniciar todas as variáveis que possam introduzir inconsistências nos comportamentos a tomar no futuro, e verificar periodicamente se existem pacotes de dados recebidos (via *ZigBee*) com novas instruções de controlo. Esta periodicidade é de 100ms.

O seguinte pseudo-código ilustra o comportamento.

```
if( state == IDLE )
{
   resetPID();
   if( newPacketReceived )
   {
      updateState();
   }
   delay(100ms);
}
```

#### 4.2.1.2. STRAIGHT

O objectivo deste comportamento é deslocar o veículo numa trajectória rectilínea (em frente). Esta tarefa é conseguida usando os *encoders* acoplados às rodas, pois sabendo o deslocamento de cada roda, calcula-se o erro e compensa-se usando um controlador PID. A taxa de actualização do PID e consequente correcção da trajectória do veículo é de 10Hz.

```
if( state == STRAIGHT )
{
   readSensors();
   // instrução que garante a periodicidade
   if( now – lastRead >= 100ms )
   {
      readEncoders();
      calcPID();
      updateMotors();
   }
   if( dataReady );
   {
      sendData();
   }
   if( newPacketReceived );
   {
      updateState();
   }
}
```

A aquisição sensorial durante este comportamento é efectuada conforme está descrito no início do parágrafo **4.2.1.**

Este comportamento pode mudar perante duas circunstâncias: quando é recebido um comando do PC com a ordem de paragem; quando é detectado um obstáculo frontal a uma distância reduzida. Em ambos os casos, o veículo robótico passa ao estado STOP.

#### 4.2.1.3. STOP

O estado de STOP é simplesmente um comportamento de transição, pois durante ele o *robot* não faz qualquer leitura de sensores, nem envia qualquer dado para o PC. Neste estado apenas é aplicado um método de travagem nos motores acoplados às rodas, por forma a imobiliza-los suave e rapidamente. Este método de travagem consiste na aplicação de uma velocidade nula durante um curto espaço de tempo aos motores, seguida de uma velocidade média (cerca de 40% da velocidade máxima) de sentido contrário durante um intervalo de tempo necessário para parar os motores. Esta inversão dos motores é necessária, uma vez que não se dispõe de travões, e o veículo seguiria em movimento devido à energia cinética acumulada, até os motores se imobilizarem por si próprios. Trata-se da mesma situação em que um condutor de um automóvel pretende parar simplesmente por retirar o pé do acelerador.

Finda a tarefa de parar, o veículo robótico passa automaticamente ao estado IDLE, à espera de novas instruções de controlo. Os dois únicos estados, nesta implementação, que podem transitar para este comportamento são logicamente o STRAIGHT e o FOLLOWALL, pois são estes que mantém o *robot* em movimento permanente.

```
if( state == STOP )
{
   driveMotors( 0, 0 );
   delay(50ms);
   driveMotors( -0.4*maxSpeed, -0.4*maxSpeed);
   delay(75ms);
   driveMotors( 0, 0 );
   delay(1000ms);

   state = IDLE;
}
```

#### 4.2.1.4. SCANNING

Este comportamento é útil em ambientes com diversos obstáculos dispersos. Assumindo que o veículo está parado, é executado um "varrimento" de 180° com os sensores de distância (partindo do princípio que estão ambos orientados na mesma direcção) – ultra-sons e infravermelhos. Esta é uma aquisição de dados mais complexa e demorada do que nos casos em que o *robot* adquire informação sensorial enquanto de desloca, porquanto aqui não existem preocupações com os tempos de amostragem no PC, mas sim que os dados sejam o mais fiáveis possível. Para tal, em cada ângulo em que os sensores são orientados, são adquiridas diversas medidas e é aplicado um filtro de média (para cada sensor). A abordagem implementada consiste num varrimento de -90° até +90° (sendo 0° a orientação frontal) com resolução de 5°, e em cada posição são adquiridos 5 valores de cada sensor. Porém, em termos de código, a programação deste comportamento é parametrizável, podendo definir o intervalo angular do varrimento, bem como a resolução e o número de medidas adquiridas em cada posição.

Como não há a preocupação com a demora da tarefa nem com os atrasos, os dados sensoriais são enviados após a aquisição em cada posição angular dos sensores. Esta implementação é adequada ao caso de os obstáculos serem estáticos. Se estes se movessem,

seria necessária uma aquisição mais rápida, para permitir que o *robot* executasse atempadamente alguma acção dependente das posições dos objectos no ambiente.

Este comportamento apenas pode ser despoletado por uma instrução de *scan* proveniente do PC quando o veículo está no estado IDLE. Finda a tarefa de recolha de dados, transita-se automaticamente para o estado IDLE novamente.

```
if( state == SCANNING )
{
   angle = -90;
   while( angle <= 90 )
   {
      readSensors();
      applyFilter();
      sendData();
      angle += 5;
   }
   state = IDLE;
}
```

Durante a execução deste comportamento o processador está-lhe reservado apenas a si, impossibilitando a execução paralela de outros comportamentos.

#### 4.2.1.5. FOLLOWALL

Este é o comportamento mais exigente e mais importante neste trabalho, pois é com base nele que é aplicado o filtro de navegação. Neste estado o *robot* segue paralelamente a uma parede, tentando sempre corrigir a trajectória medindo a distância à parede e contrariando o erro. Esta tarefa é conseguida usando um sensor de distância direccionado perpendicularmente à direcção de deslocamento do veículo (ou seja, virado para a parede), que mede constantemente a distância à parede.

O sensor de distância utilizado para medir as distâncias à parede é o IR, pois tipicamente pretende-se seguir próximo da parede (20cm), e este sensor tem elevada precisão para curtas distâncias, além de ter uma resposta mais rápida que o sensor de ultra-sons.

```
if( state == FOLLOWALL )
{
   readSensors();
   // instrução que garante a periodicidade
   if( now – lastUpdate >= 100ms )
   {
      calcPID();
      updateMotors();
   }
   if( dataReady );
   {
      sendData();
   }
   if( newPacketReceived );
   {
      updateState();
   }
}
```

O objectivo deste comportamento é manter uma distância predefinida, denominada de **distância de referência**, a algo existente ao lado do veículo, e não somente seguir uma parede. Portanto este comportamento aplica-se a paredes (ou outras superfícies verticais contínuas) planas, côncavas ou convexas (ver **Figura 4.6**).

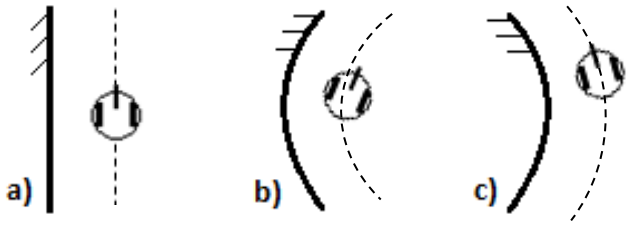



**Figura 4.6** – *Robot* no seguimento de parede: **a)** plana; **b)** côncava; **c)** convexa.

A correcção da distância é feita aplicando um controlador PID, tal como no caso do comportamento STRAIGHT. Porém, aqui o PID não corrige o erro entre o deslocamento de cada roda, mas sim a distância medida à parede em comparação com a distância de referência. Como o período mínimo de aquisição de dados do sensor de distância em causa é de 50ms, é utilizada a média entre duas medidas como distância actual à parede a ser corrigida pelo PID. Desta forma, suaviza-se a trajectória do veículo por redução do erro associado às medidas, e tem-se uma taxa de actualização do PID de 10Hz, analogamente ao que acontece no comportamento STRAIGHT.

A particularidade deste comportamento reside no facto de se pretender que a trajectória do veículo seja aproximadamente imune às variações bruscas da parede, tais como as reentrâncias das portas. Pretende-se que o *robot* siga a parede tomada como referência global mas que não tente seguir reentrâncias na mesma, tal como vãos das portas, fendas, etc. Todas as medidas consecutivas adquiridas pelo sensor de distância são comparadas, e isso permite detectar quando há variações bruscas, que são interpretadas

como reentrâncias. Quando tal é detectado, a distância de referência é actualizada temporariamente para a nova distância actual à parede (notar que esta é a média entre duas medidas), de forma a garantir que o veículo mantém a sua trajectória aproximadamente rectilínea. Esta situação está ilustrada na Figura 4.7. Em termos práticos, a distância de referência, que é constante, é traduzida no *set_point* do PID. Este é ajustado ao longo do tempo consoante o percurso o impõe.

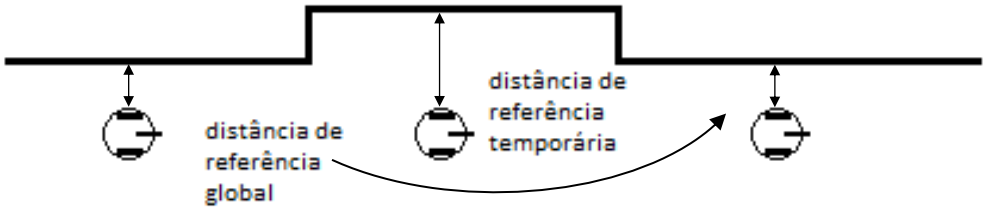



**Figura 4.7** – Ilustração da distância de referência e sua actualização ao longo de uma parede.

Isto levanta um problema: se houver sistematicamente erros associados à medição da distância de cada vez que o *set_point* é ajustado, o veículo poderá afastar-se ou aproximar-se demasiado da parede, sendo o segundo cenário o mais indesejado. Este problema é contornado utilizando duas possíveis soluções:

- De cada vez que uma reentrância é detectada, o *set_point* é comparado com a distância de referência, e caso este seja demasiado inferior, é-lhe somado um pequeno valor (no caso implementado, um centímetro), para não interferir demasiado com o PID. Esta solução mantém o veículo numa trajectória razoavelmente suave, previne que se aproxime muito da parede, mas não evita que o *set_point* oscile ao longo do tempo;

```
loop( readDistance )
{
   if( ledgeDetected )
   {
      if( set_point < 165mm )
      {
         set_point = set_point + 10mm;
      }
   }
}
```

- O *set_point* actual é constantemente comparado com a distância de referência, e caso seja mais de um centímetro inferior, são-lhe somados cinco milímetros. Esta solução introduz ligeiramente mais oscilações no trajecto do veículo, pois interfere constantemente com o PID, mas a longo prazo mantém a distância do *robot* à parede mais próxima da distância de referência predefinida.

```
loop( readDistance )
{
   if( set_point < 190mm )
   {
     set_point = set_point + 5mm;
   }
}
```

Ambas as soluções foram implementadas e testadas no controlo de trajectórias, conforme é apresentado no Capítulo **5**, O controlo da trajectória no percurso usado no teste de navegação foi implementado com a segunda solução. Os resultados são apresentados no Capítulo **6**.

A aquisição e envio dos dados sensoriais são feitos conforme foi explicado na subsecção **4.2.1**.

Este comportamento só pode ser executado a partir do estado IDLE, através de um comando enviado pelo PC. Por sua vez, este comportamento só pode passar para o estado IDLE, indirectamente pelo estado STOP. As duas situações que podem activar esta transição de estado são:

- Quando é enviada, a partir do PC, uma instrução de paragem do veículo;
- Quando deixa de ser detectada a parede que está a ser seguida (distância superior a 60cm).

#### 4.2.1.6. TURN

Este comportamento é bastante simples: o veículo efectua uma rotação de 90° sobre uma roda com base na informação de um dos *encoders*. Sabendo as características das rodas e a distância entre as rodas, é fácil calcular quantos impulsos são gerados pelos *encoders* numa rotação de 90°. Esta rotação faz-se sobre uma roda porque tem-se em conta os erros e a resolução dos *encoders*. Numa rotação com as duas rodas (sobre o próprio eixo), existem duas fontes de erro, e se ambas as rodas andarem o mesmo número de contagens, a resolução passa a metade – por cada contagem do *encoder*, cada roda desloca-se cerca de meio centímetro. Assim, rodando sobre apenas uma roda, a rotação é feita em passos de meio centímetro (em vez do dobro), e o erro de um *encoder* não afecta tanto a precisão da rotação.

Na implementação actual, este comportamento apenas roda o veículo 90° à sua direita. Porém, facilmente se adaptaria para o caso geral, dado que o princípio é o mesmo. Apenas se implementou esta rotação em particular porque no cenário de teste à navegação, o veículo apenas precisa de rodar 90° à sua direita.

Durante este comportamento, não são trocadas quaisquer informações com o PC, e os únicos dados sensoriais envolvidos são os valores do *encoder* da roda que se desloca.

Quando o veículo está no estado IDLE e recebe uma instrução para rotação, transita para este comportamento, retornando a IDLE quando termina a rotação. Note-se que aqui não há passagem pelo estado STOP, pois o modo de travagem é diferente.

```
if( state == TURN )
{
   while( leftEncoder < nPulses )
   {
      driveMotors( 0.3*maxSpeed, 0 );
   }
   stopWheels();
   state = IDLE;
}
```

#### 4.2.1.7. ANGLE

Este comportamento também engloba uma rotação, mas ao passo que no estado TURN se trata de uma orientação relativa, aqui a orientação é global. Aqui, recorre-se às medidas de orientação adquiridas pela bússola digital, para direccionar o veículo num ângulo desejado entre 0° e 359°, em que 0° corresponde ao Norte.

```
if( state == ANGLE )
{
   while( | currentAngle – desiredAngle | > tolerance)
   {
      turnVehicle();
      currentAngle = readCompass();
   }
   stopWheels();
   state = IDLE;
}
```

Neste estado, não há qualquer troca de dados entre o *robot* e o PC, nem são adquiridos quaisquer dados sensoriais para além dos obtidos pela bússola. Quando se recebe do PC a instrução de rotação com base na bússola, incluindo o ângulo desejado, o veículo robótico vai rodando sobre o seu eixo e lendo a orientação actual até esta coincidir com a orientação global desejada, com uma tolerância atribuída de 2°. A orientação actual é obtida a partir da média entre um número de medidas adquiridas. Este número tem de ser baixo, pois o veículo está em rotação ao mesmo tempo que adquire os dados, e portanto pode passar o ângulo desejado enquanto a bússola está a recolher dados, e não parar a tempo. No entanto, este filtro de média é necessário para reduzir os erros, pelo que o número de medidas utilizado é três.

Este comportamento de rotação tem a desvantagem de que a bússola é mais sensível a interferências externas que influenciem o campo magnético, pelo que o veículo pode ficar orientado erraticamente. Por outro lado, em relação à rotação com base nos *encoders*, este comportamento apresenta a vantagem de ter uma referência fixa, o que providencia uma orientação global do veículo. Os comportamentos ANGLE e TURN complementam-se bastante bem, pois enquanto que um é global mas é sensível a interferências externas, o outro é relativo e é mais robusto em termos de erro.

Analogamente ao estado TURN, o estado IDLE pode transitar para ANGLE ao receber a respectiva instrução proveniente do PC. Quando a rotação é terminada, o veículo passa ao estado IDLE, voltando a esperar por novas instruções.

## 4.3. Algoritmo de Navegação

O algoritmo de navegação utilizado no trabalho foi desenvolvido pelo meu orientador Prof. Francisco Curado, e corre no Simulink (em *Matlab*), e basicamente tem como finalidade representar num mapa pré adquirido a localização estimada de um veículo robótico (ou outro agente munido de sensores adequados).

Neste trabalho, o problema principal a resolver é a localização do veículo em 2D. Duas boas soluções para este problema, apresentadas em **[30]**, são os algoritmos: Localização com base numa Grelha de Ocupâncias, e Localização de Monte Carlo. Estes algoritmos conseguem processar dados brutos provenientes de sensores, e conseguem resolver os problemas da localização global e do *robot* "raptado"[8].

No primeiro caso, o mapa é representado por uma grelha de ocupação, ou seja o mapa é dividido numa grelha de células. A cada célula está associada uma probabilidade de estar ocupada pelo *robot* ou por um obstáculo. Além das probabilidades, a estimativa da posição do *robot* pode também ser representada por um histograma distribuído pelas células.

O algoritmo da Localização de Monte Carlo é o mais comum, e é também o utilizado pelo algoritmo de navegação neste trabalho, implementado num Filtro de Partículas (FP).

Um filtro de partículas é um algoritmo que implementa um método de Monte Carlo (MC) sequencial com o objectivo de estimar a densidade de probabilidade associada ao espaço de estados de um sistema usando métodos de estimação Bayesiana[9] **[31, 32]**. Num FP, tal como num processo MC clássico, em cada iteração é gerado aleatoriamente um novo conjunto finito de estados hipotéticos, designados 'partículas', aos quais são atribuídos pesos calculados com base numa função de verosimilhança. As partículas são geradas usando um modelo estocástico da dinâmica do sistema, que determina como os estados evoluem de uma iteração para a seguinte e os pesos são atribuídos calculando a função de verosimilhança dos respectivos estados tendo em conta as medições reais efectuadas pelo sistema nessa iteração. Estas duas componentes do cálculo de um filtro de partículas – geração de novas partículas e cálculo dos pesos – correspondem a dois passos típicos do processamento executado por um filtro de partículas em cada iteração: Predição e Filtragem. Em dado momento (iteração do filtro), a distribuição espacial das partículas conjugada com os respectivos pesos representam a distribuição de probabilidade dos estados do sistema. Uma estimativa pontual do estado do sistema em dado momento pode ser obtida aplicando, por exemplo, uma média pesada de todas as partículas geradas pelo filtro.

Uma das maiores vantagens deste tipo de filtros é a sua capacidade de representar e processar distribuições de probabilidade arbitrárias e de lidar naturalmente com modelos não lineares quer da dinâmica do sistema quer das observações.

Na corrente aplicação em robótica móvel, o modelo da dinâmica usado pelo filtro de partículas é bastante simples, correspondendo ao modelo cinemático do veículo, e explora

[8] Problema típico de robótico em que um *robot* é instantaneamente retirado da sua posição actual e colocado numa outra posição do espaço, sem que haja uma transição contínua nesta movimentação.
[9] Na estimação Bayesiana, estima-se a densidade de probabilidade *a posteriori* de um estado, assumindo que existe sempre informação *a priori* sobre esse mesmo estado, a qual é aumentada através da incorporação de nova informação.

os dados de odometria para prever a evolução do estado (pose) do robot de uma iteração para a seguinte ao executar o passo de Predição. As observações usadas no passo de Filtragem correspondem às medições das distâncias realizadas pelo veículo em cada iteração; estas são comparadas com as medições previstas pelo filtro, tendo como base um mapa do ambiente, para calcular a verosimilhança de cada partícula nessa iteração. Este cálculo da função de verosimilhança baseia-se na simulação das observações esperadas para cada partícula, o que implica a utilização de um modelo estocástico das medições que pode ser relativamente complexo dada a natureza dos sensores utilizados e os erros associados às respectivas medições. Além disso, uma vez que são usados sensores distintos para o mesmo fim, a filtragem implica a utilização de técnicas de fusão sensorial adequadas que devem ser integradas no filtro.

Na **Figura 4.8**. estão ilustradas duas possíveis distribuições das partículas pelo mapa. Note-se que apesar de a nuvem de partículas ser dispersa por uma vasta área, as partículas mais afastadas têm pesos praticamente nulos, estando as de maior peso localizadas em torno da localização mais provável do *robot*.

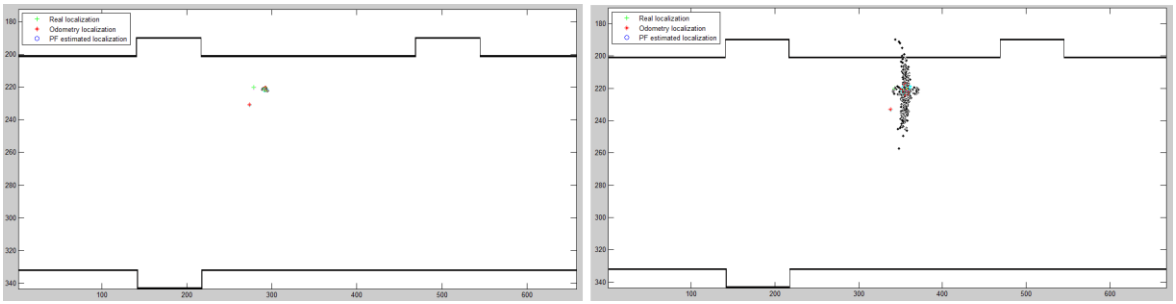



**Figura 4.8** – Exemplos de distribuições de partículas no algoritmo de navegação.

## 4.4. Mapeamento/Obtenção do Mapa

O cenário onde foram corridos os testes para aplicação do algoritmo de navegação é bastante simples. O ambiente escolhido foi um corredor do 3º piso do Departamento de Geociências da Universidade de Aveiro, por ter várias portas de ambos lados ao longo das paredes de cerca de 30 metros. Este cenário é conveniente, pois é fácil projectar e concretizar um trajecto para o veículo robótico, tem bastantes pontos de referência para situar o *robot* aquando da aplicação do algoritmo de navegação, e é fácil de mapear em *Matlab*.

Este ambiente tem no entanto alguns pontos fracos para efeitos de navegação. As portas ao longo do corredor estão espaçadas regularmente e de forma quase simétrica relativamente ao eixo do corredor, o que dificulta a distinção entre localizações equiprováveis no algoritmo de navegação. Este é o primeiro cenário, em que nenhuma porta foi ocultada e a simetria é bastante notada (ver Figura A.1).

No segundo cenário, foram ocultadas duas portas (não simétricas) no corredor, eliminando significativamente a simetria do mapa, como pode ser observado na Figura A.2.

O controlo de trajectórias foi executado nestes dois cenários, com os resultados analisados na subsecção **5.6**. No Capítulo 6 são apresentados os resultados do algoritmo de navegação com os dados adquiridos pelo *robot* no segundo cenário.

Para a obtenção do mapa, foram recolhidas manualmente, com precisão de 0,5cm, todas as medidas relevantes no corredor, tais como distâncias entre as portas e as medidas (largura e profundidade) das suas reentrâncias. Com esta informação completa do corredor, este é representado no *Matlab* como uma matriz binária, em que o valor 1 representa a existência de parede, e o valor 0 o contrário. Devido às precisões dos sensores de distância e para não sobrecarregar o algoritmo de navegação, que tem de efectuar cálculos probabilísticos para grande parte do espaço do mapa, decidiu-se que a resolução ideal para o mapa é de 1cm. Desta forma, um mapa quadrado com um metro de lado será representado por uma matriz de 100 linhas e 100 colunas.

O corredor termina numa sala ampla em cada uma das extremidades. Porém, estas não eram relevantes para nenhum teste e portanto foram não representadas.

## 4.5. Controlo via *Matlab*

Para comandar o veículo robótico a partir do PC, foi desenvolvido um sistema de controlo e monitorização no *Matlab*. Este sistema consiste num interface gráfico com o utilizador (GUI) em *Matlab* que recebe em tempo-real os dados provenientes do veículo robótico, identifica-os em termos dos respectivos sensores, processa-os caso seja necessário, e representa-os sob a forma de um gráfico para melhor análise pelo ser humano. Além da recepção, o GUI disponibiliza ainda os comandos necessários para fazer o *robot* transitar entre comportamentos de forma interactiva com o utilizador, e assim executar as tarefas desejadas pelo utilizador, tais como andar frente, parar e rodar.

Com este GUI, na Figura 4.9, é assim possível ver em tempo-real o que cada sensor do *robot* está a adquirir, bem como comandá-lo para executar outros comportamentos. No **Anexo B** é apresentada a legenda do interface gráfico, com a descrição de cada elemento.

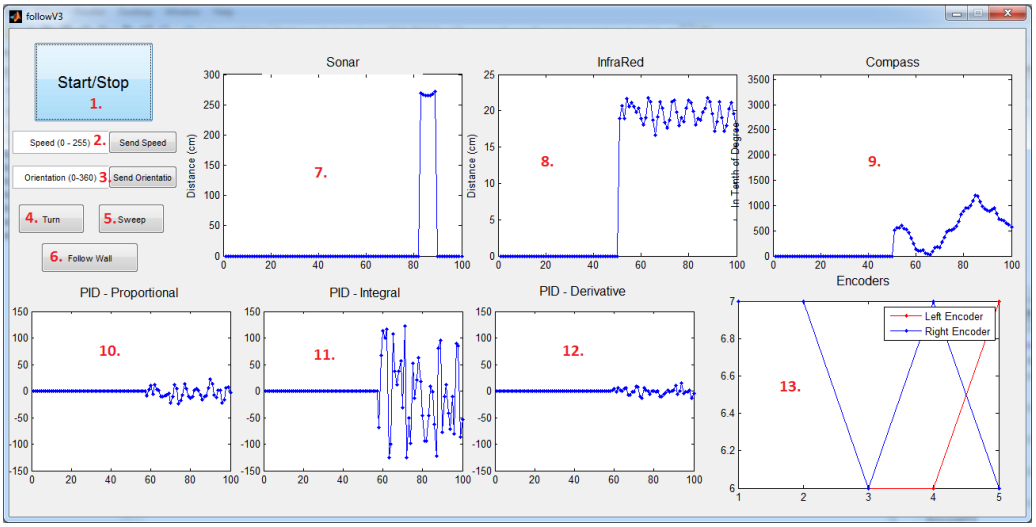



**Figura 4.9** – Interface Gráfico em *Matlab*.

### 4.5.1. Estrutura dos Pacotes de Dados

Para permitir a monitorização dos dados sensoriais e controlo do veículo robótico, são trocados pacotes de dados, através dos módulos *XBee*, em ambos os sentidos. Para o efeito, foi criada uma estrutura dentro do campo *Data* das tramas *ZigBee*. Esta estrutura também depende do sentido de envio do pacote de dados.

#### 4.5.1.1. *Robot* -> PC

No sentido do *robot* para o PC, são enviados três tipos de pacotes de dados:

- Informação dos sensores de distância e bússola;
- Informação dos *encoders*;
- Erros associados aos três componentes do PID.

Este terceiro tipo de pacote de dados só é relevante e usado na fase de afinação dos parâmetros do controlador PID; quando este está com os parâmetros adequados, não é enviada a informação dos seus componentes, por forma a tornar o sistema mais rápido. Note-se ainda que o envio dos dados do PID apenas faz sentido nos comportamentos STRAIGHT e FOLLOWALL, pois apenas estes utilizam o PID para controlo do movimento (de trajectória ou distância à parede, respectivamente).

A periodicidade do envio dos pacotes foi explicada e justificada na secção **4.2.** Esta abordagem é seguida para os três tipos de pacotes de dados enviados para o PC. O código do *Arduino* está generalizado, isto é, o valor dos parâmetros de periodicidade de amostragem e número de medidas enviadas por pacote de dados podem ser facilmente alterados conforme o desejado. No presente caso, todos os pacotes de dados são enviados com cinco medidas, amostradas com periodicidade de 100ms.

Os três tipos de dados existentes são organizados no campo de dados de uma trama *ZigBee* da seguinte maneira:

- Pacote de dados dos sensores[10]

| Identificador | *Time Tag* | $IR_1$ | $Sonar_1$ | $Bússola_1$ | … | $IR_N$ | $Sonar_N$ | $Bússola_N$ |
|---|---|---|---|---|---|---|---|---|
| 0x0a | 4 bytes | 2 bytes | 2 bytes | 2 bytes | … | 2 bytes | 2 bytes | 2 bytes |

Em todos os casos os dados enviados são representados por dois bytes porque contêm valores sempre inferiores a $2^{16}$. Já no caso da *tag* temporal, este contém um valor *unsigned long*, isto é, um valor inteiro positivo de 32bit, necessitando assim de 4bytes para o representar. O campo inicial tem apenas a finalidade de identificar o tipo de dados contidos no pacote, e como apenas existem 3 tipos diferentes (nesta implementação do trabalho), um byte é o suficiente.

[10] Por "sensores", referem-se os IR, ultra-sons e bússola digital.

- Pacote de dados dos *encoders*

| Identificador | *Time Tag* | *Left Encoder*$_1$ | *Right Encoder*$_1$ | … | *Left Encoder*$_N$ | *Right Encoder*$_N$ |
|---|---|---|---|---|---|---|
| 0x0b | 4 bytes | 1 byte | 1 byte | … | 1 byte | 1 byte |

Tal como anteriormente, o primeiro campo identifica o tipo de dados, mas agora com um valor diferente. A *tag* temporal tem exactamente a mesma funcionalidade em todos os tipos de dados, pelo que a explicação dos 4bytes é a mesma. Finalmente, como o veículo anda relativamente devagar e os *encoders* são lidos frequentemente (e repostos a zero), os seus valores nunca chegam a ser superiores a $2^7$ (um bit é para o sinal), pelo que um byte é o suficiente para os representar.

- Pacote de dados dos erros associados ao PID

| Identificador | *Time Tag* | $P_1$ | $I_1$ | $D_1$ | … | $P_N$ | $I_N$ | $D_N$ |
|---|---|---|---|---|---|---|---|---|
| 0x0c | 4 bytes | 1 byte | 1 byte | 1 byte | … | 1 byte | 1 byte | 1 byte |

O identificador e o "selo" temporal são também aqui utilizados, pelas mesmas razões. Cada um dos três componentes do controlador PID é representado por um byte porque o erro associado a cada é tipicamente pequeno, mesmo quando representado em milímetros. Principalmente quando o PID já está ajustado, estes erros são da ordem de um ou dois centímetros tipicamente, sejam eles de natureza proporcional, integral ou derivativa.

Note-se que em todos os casos a letra N representa um valor genérico. Na implementação actual do sistema, este número é 5 para todos os casos.

Em termos de tempos de amostragem, existem três pontos de vista: do *Arduino*, que adquire e processa em primeira mão todos os dados dos sensores a si ligados; do PC, em tempo-real; e do PC, *offline*.

No *Arduino*, existem diferentes tempos de amostragem para os diferentes sensores, o que é algo problemático, pois para facilitar a implementação dos algoritmos de navegação, pretende-se ter medidas síncronas de todos os sensores. É por isso que a cada 100ms são guardados apenas os dados mais recentes de cada sensor. O sensor de distância IR é lido a cada 50ms, o sensor de distância ultra-sons a cada 70ms e a bússola digital a cada 7ms. Esta última, por ter uma taxa de aquisição muito elevada, cada medida de orientação utilizada (isto é, que é enviada para o PC) é na verdade a média entre dez leituras consecutivas. Os períodos de leitura indicados são os tempos mínimos aconselhados pelos respectivos fabricantes. A periodicidade de leitura dos *encoders* (que é a mesma que a actualização do PID) é de 100ms, conforme foi explicado na secção **3.6**.

No capítulo seguinte, na secção **5.8.1** é discutida a noção de tempo-real da amostragem de dados, do ponto de vista de cada um dos casos: *Arduino*, GUI no PC, e algoritmo de navegação no PC.

#### 4.5.1.2. PC -> *Robot*

No sentido "PC -> *Robot*" da transmissão de dados, a temporização já não é tão crítica, pois são enviados os comandos de controlo de forma assíncrona, quando a entidade de decisão do comportamento acha apropriado. Na actual implementação do trabalho, quando o utilizador pretende enviar um comando ao veículo robótico, fá-lo interagindo nesse sentido através do GUI. Este constrói a trama *ZigBee* completa (correspondente ao comando a enviar) e envia-a ao *robot*. Porém, este cenário pode não ser aplicável num sistema de controlo e navegação real, pois se o ambiente for muito complexo e implicar frequentes mudanças de estado (que são comandadas pelo PC), as comunicações poderão ficar sobrecarregadas, introduzindo atrasos críticos para o comando do *robot*. No entanto, este é, de facto, o objectivo: navegação em tempo real.

Apesar do veículo robótico poder desempenhar sete comportamentos, foram implementados cinco tipos de comandos. O comportamento STOP consiste numa transição para o estado IDLE. Portanto, o estado IDLE pode ser obtido indirectamente através de um comando STOP. Por outro lado, o comando STOP consiste no envio de um comando STRAIGHT cuja velocidade especificada é zero. Desta maneira, omite-se um tipo de comando para o estado IDLE através do comando STOP, que por sua vez é integrado num caso particular do comando STRAIGHT.

O tipo de comando é especificado na organização do campo *Data* da trama *ZigBee*. Os tipos de comandos possíveis de enviar do PC para o veículo robótico são os apresentados na tabela abaixo, em que o nome de cada pacote é o estado para o qual o *robot* transita quando o recebe.

**Tabela 4.2** – Tipos de comandos enviados do PC para o *robot*.

| Pacote | Tipo de comando | Parâmetros |
|---|---|---|
| STOP | 0x01 | Velocidade = 0 |
| STRAIGHT | 0x01 | Velocidade desejada (1 – 255) |
| FOLLOWALL | 0x02 | Não aplicável (actualmente) |
| ANGLE | 0x03 | Orientação global desejada (0° – 359°) |
| TURN | 0x04 | Não aplicável (actualmente) |
| SCAN | 0x05 | Não aplicável (actualmente) |

Os pacotes apresentados acima necessitam apenas do identificador, pois na maioria dos casos, perante a sua recepção, o veículo robótico apenas transita para o estado correspondente. As únicas excepções são os pacotes IDLE, STRAIGHT e ANGLE, que precisam de especificar a velocidade pretendida para o veículo, ou qual o ângulo segundo o qual o *robot* se deve orientar.

Uma melhoria no futuro será enviar os parâmetros associados a cada comportamento juntamente com o comando, nomeadamente nos comandos TURN, FOLLOWALL ou mesmo SCAN.

# 5. Implementação do Veículo Robótico e Integração no Sistema de Navegação Cooperativa

## 5.1. Introdução

### 5.1.1. Robots

Existem diversas definições de *robot*. Podem indicar-se algumas como:

- "Eu não consigo definir o que é um *robot*. Mas consigo identifica-lo, quando vejo um." – Engelberger, o "pai da robótica industrial" **[33]**
- Dispositivo electromecânico capaz de desempenhar tarefas por si só, ou com orientação;
- Agente físico que cria uma ligação inteligente/autónoma entre percepção e acção;
- Sistema autónomo no mundo físico capaz de "sentir" o ambiente e actuar sobre ele de modo a atingir um conjunto de objectivos.

As duas grandes áreas da robótica são a Robótica Móvel e a Robótica Industrial. Na robótica móvel, a maior expansão e evolução encontra-se ao nível da robótica móvel subaquática. Neste trabalho de mestrado, o objectivo é estudar um veículo robótico móvel em cenários *indoor*.

Na robótica móvel, é incontornável a referência aos veículos robóticos Mars Rovers, que são um grupo de *robots* desenvolvidos pelo *Jet Propulsion Laboratory* da NASA com o objectivo principal de vaguear pela superfície de Marte **[34]**. Recolhem imagens e outras amostras geológicas do ambiente marciano, de modo a conhecer o passado de Marte e também do nosso planeta; objectiva-se ainda estudar a possibilidade de sobrevivência humana em Marte. Foram colocados naquele planeta quatro veículos robóticos com sucesso:

- Sojourner (1997)
- Spirit (2004 – 2010)
- Opportunity (2004 – presente)
- Curiosity (2012 – presente)

### 5.1.2. Navegação Cooperativa

A navegação cooperativa ocorre quando em vez de um, temos vários *robots* (dezenas, por exemplo) que comunicam entre si com o objectivo de colaborarem na execução de uma missão. Este conceito é inspirado em parte no comportamento dos animais, como por exemplo as formigas, que na natureza trabalham em grupo para atingir um objectivo comum.

Em **[35]**, os *robots* (muito básicos) executam um algoritmo de navegação cooperativa que requer comunicação entre eles. Usando trocas de mensagens de localização, ajudam um único *robot* a chegar a um ponto destino, ou a viajarem em grupo repetidamente entre dois pontos. Este segundo cenário é muito parecido ao que acontece com as formigas, que ao deslocarem-se deixam feromonas no ar que ajudam outras a seguir o trajecto correcto.

### 5.1.3. Ferramentas úteis em Robótica – ROS

Actualmente, uma ferramenta muito utilizada em robótica e que ganha cada vez mais "adeptos", é o ROS – *Robot Operating System*. Trata-se de um sistema operativo com uma arquitectura distribuída que está preparado para atribuir tarefas às diferentes unidades de processamento que estiverem disponíveis **[36] [37]**. Isto potencia a implementação de tarefas bastante complexas, como a navegação robótica multi-agente em tempo real. Além disso, tratando-se de uma ferramenta *open source*, a quantidade de informação disponível e partilhada é muito vasta, e cresce de dia para dia.

Inicialmente ponderou-se a ideia de implementar o projecto em ROS, o que seria muito interessante. Porém, devido a limitações de tempo e à imensa aprendizagem necessária para dominar o ROS, optou-se por implementar em *Matlab*, ferramenta na qual os conhecimentos eram já consolidados. No entanto, uma implementação em ROS seria muito proveitosa, e sem dúvida que seria algo a seguir como trabalho futuro.

### 5.1.4. *Hardware* integrado no Veículo Robótico

No veículo robótico construído neste projecto, estão integrados os seguintes elementos:

- Um kit *Arduino* (*Duemilanove*);
- Um *screw shield* para aplicar ao *Arduino* (para facilitar as ligações de cabos de sinal);
- Um *shield* para comando e alimentação de motores, também para *Arduino*;
- Dois sensores de distância: um infravermelhos e um ultra-sons;
- Uma bússola digital;
- Duas rodas, às quais são acoplados dois motores DC e os respectivos *encoders*;
- Dois rádios *XBee* de comunicação *ZigBee* (um para o *Arduino*, um para o PC);
- Um servomotor (para orientação dos sensores de distância);

- Dois suportes para conjuntos de 8 pilhas recarregáveis do tipo AA, para alimentar independentemente o *Arduino* e o *shield* de motores;
- Uma roda livre ( "*caster ball*") para oferecer o terceiro apoio ao veículo robótico;
- Chassis, suportes para os componentes e cablagem.

Dividindo as tarefas entre *Arduino* e PC, cada um executa uma parte das tarefas do sistema. O *Arduino* corre um código que trata da recepção e execução dos comandos enviados pelo PC, a aquisição sensorial, o envio de dados para o PC e o comando dos motores. No PC corre em *Matlab* um *script* que fornece um interface gráfico, onde é possível fornecer os comandos para controlo do *robot*, bem como visualizar os dados sensoriais que vão sendo recebidos, para serem usados no algoritmo de navegação.

## 5.2. *Arduino*

O *Arduino* é o cérebro presente no veículo robótico, responsável por receber dados sensoriais, actuar nos motores, e trocar dados com o PC.

O fabricante oficial define o *Arduino* como "(…) uma plataforma *open-source* para prototipagem em electrónica, baseada em *hardware* e *software* flexível e fácil de usar. É direccionado para artistas, designers, *hobbyists*, e qualquer utilizador interessado em criar objectos ou ambientes interactivos" **[38]**.

Existem diversas versões de *Arduino*, com diferentes tamanhos, propósitos e especificações técnicas, bem como inúmeros *shields*. Nesta dissertação, foram utilizados apenas dois tipos de *Arduino*: *Duemilanove* e *Uno*. Estes dois são bastante parecidos, até porque se tratam de lançamentos consecutivos do mesmo modelo do *Arduino*, sendo o *Uno* o mais recente. As suas especificações técnicas são as presentes na **Tabela 5.1**.

**Tabela 5.1** – Características técnicas do *Arduino Duemilanove* e *Uno*.

| **Característica** | ***Arduino Duemilanove*** | ***Arduino Uno*** |
|---|---|---|
| Microcontrolador | ATmega328 | ATmega328 |
| Tensão de Funcionamento | 5V | 5V |
| Tensão de Alimentação (Recomendado) | 7-12V | 7-12V |
| Tensão de Alimentação (Limites) | 6-20V | 6-20V |
| Pinos I/O Digitais | 14 (dos quais 6 podem fornecer PWM) | 14 (dos quais 6 podem fornecer PWM) |
| Pinos Entrada Analógicos | 6 | 6 |
| Corrente DC por pino I/O | 40mA | 40mA |
| Corrente DC por pino 3.3V | 50mA | 50mA |
| Memória Flash | 32KB, 2KB são usados pelo bootloader | 32KB, 0.5KB são usados pelo bootloader |
| SRAM | 2KB | 2KB |
| EEPROM | 1KB | 1KB |
| *Clock Speed* | 16MHz | 16MHz |

Como se pode verificar acima, os dois modelos de *Arduino* são muito semelhantes, apenas diferindo na memória usada pelo *bootloader*.

Tal como foi referido, a utilização do *Arduino* no veículo robótico é justificada pela sua grande versatilidade, facilidade em ligar *hardware*, simplicidade na utilização do ambiente de desenvolvimento de *software*, e principalmente pelo facto de ser uma plataforma *open-source*. Tem uma quantidade razoável de pinos digitais que podem servir como entradas ou saídas, dos quais alguns podem operar como saídas de PWM (usados para controlar motoros, por exemplo). Está equipado também com seis entradas analógicas (muito úteis para diversas aplicações, como por exemplo ler o sensor de infravermelhos utilizado no trabalho – ver secção **3.3.2**). Devido à sua vasta utilização, quer em termos de quantidade de utilizadores, quer em tipos de aplicações, existe uma elevada quantidade de documentação, trabalho desenvolvido e comunidades de apoio como fóruns. Todos estes factores potenciam o projecto, pois a reutilização de ideias e trabalhos evitam o reinventar de algo que já está feito e documentado.

O *Arduino* possui também dois *interrupts* externos, e dois pinos que permitem a comunicação com outro dispositivo segundo a norma I2C. Tem ainda um led embutido, que facilita o processo de *debug* (quando não é possível recorrer ao monitor série), e ainda a interpretar o comportamento do *Arduino*.

Mais um argumento importante a favor da utilização do *Arduino* é o facto de existir uma grande quantidade de *shields* próprios para as mais diversas aplicações, tais como comunicações, controlo de motores ou *displays* LCD. Alguns *shields* podem ainda ter bibliotecas associadas, o que facilita ainda mais a interacção com eles.

Existem soluções alternativas aos modelos de *Arduino* utilizados. Dentro das opções disponíveis do *Arduino*, são de salientar o *Mega* e o *Due*, que são os mais poderosos e recentes, sendo que o *Due* se destaca por ser o primeiro da marca baseado num microcontrolador ARM de 32 *bit*. O *Arduino* comercializa também o *Arduino Robot*, que é um *robot* com dois processadores – um associado ao controlo dos motores e outro associado aos sensores e ao modo de operação do *robot* –, sendo cada um programável através do IDE.

Outras alternativas ao *Arduino* consistem na utilização de mini PC's. O mais conhecido é o recente *Raspberry Pi* **[39]**, que é um computador do tamanho de um cartão de crédito, no qual todo o *hardware* é integrado numa placa única. Corre um sistema operativo, tipicamente o *Raspbian*, baseado em *Linux*. Este computador tem uma capacidade de processamento e de memória muito superior ao *Arduino*, o que possibilita a implementação de tarefas mais complexas. Porém não dispõe das portas digitais e analógicas que estão disponíveis no *Arduino*.

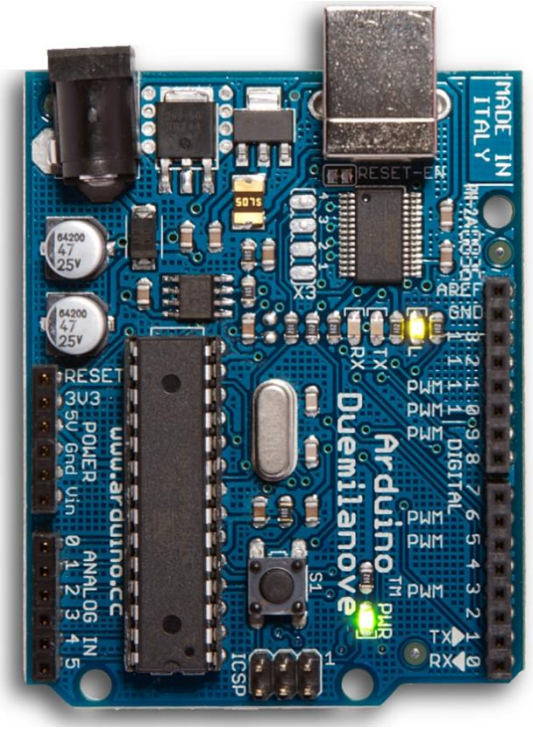
Um último dispositivo relevante de referir é o projecto *UDOO*, um *single-board computer* que combina um processador ARM *quadcore* com um *Arduino*, *Android* e *Linux* no mesmo dispositivo **[40]**. Trata-se de uma plataforma desenhada para fornecer um ambiente flexível que permita explorar as novas fronteiras da "Internet das Coisas". Pode correr *Linux* (*Linaro*, mais concretamente), *Android* e é compatível com todos os ficheiros e *shields* existentes para *Arduino* (até ao *Due*). Os fabricantes iniciaram a distribuição do produto em Outubro de 2013.

Para potenciar a capacidade do *Arduino*, foram utilizados dois *shields* acoplados a este. O "*ScrewShield*" possibilita a conexão de fios multifilares ao *Arduino*, porque estende

todas as portas para terminais de parafuso de 3.5mm, mantendo também as originais. Além disso, dispõe de uma placa de circuito impresso com furação que permite a montagem de pequenos circuitos ligados às portas do *Arduino*. No caso deste trabalho foram montadas no *shield* duas resistências de "*pull-up*" necessárias para a comunicação I2C do *Arduino* com mais do que um dispositivo. O segundo *shield* utilizado é um *driver* para controlo de motores da Adafruit. Este *shield* está preparado para que lhe sejam conectados dois servomotores, e quatro motores DC ou dois motores de passo. Neste trabalho são ligados ao *shield*: um servomotor e dois motores DC.

Apresenta-se na **Tabela 5.2** a listagem das funções associadas às portas do *Arduino*, quando todos os componentes do *robot* estão conectados.

**Tabela 5.2** – Listagem das portas do *Arduino* e suas funções.

| | |
|---|---|
|  | D0 (Rx) – Porta $D_{OUT}$ do módulo *XBee*<br>D1 (Tx) – Porta $D_{IN}$ do módulo *XBee*<br>D2 (int0) – Leitura do *encoder* da roda esquerda<br>D3 (int1) – Leitura do *encoder* da roda direita<br>D4 – Reservado ao *shield* de *drive* de motores<br>D5 – Utilizado pelo *shield* para activação e controlo de velocidade do motor 3 (roda direita)<br>D6 – Utilizado pelo *shield* para activação e controlo de velocidade do motor 4 (roda esquerda)<br>D7 – Reservado ao *shield* de *drive* de motores<br>D8 – Reservado ao *shield* de *drive* de motores<br>D9 – Utilizado pelo *shield* para controlo do motor servo nº 2<br>D10 – Leitura do *encoder* da roda direita (na saída que não está conectada ao pino D3)<br>D11 – Leitura do *encoder* da roda esquerda (na saída que não está conectada ao pino D2)<br>D12 – Reservado ao *shield* de *drive* de motores<br>A0 – Pino $V_{OUT}$ do sensor de distância infravermelho<br>A4 – Sinal SDA (barramento I2C) da bússola digital e também do sensor de distância de ultra-sons<br>A5 – Sinal SCL (barramento I2C) da bússola digital e também do sensor de distância ultra-sons |

## 5.3. O Veículo Robótico – "*GEO-Rover*"

Nesta dissertação, pretendia-se que o veículo robótico fosse capaz de obter dados sensoriais, comunicá-los ao PC e receber deste comandos de forma a interagir com o meio. Os dados sensoriais adquiridos pelo *robot* são:

- Medidas de distância a obstáculos;
- Orientações fornecidas pela bússola digital;
- Contagens de rotações das rodas.

O esquema eléctrico completo do veículo robótico é apresentado na Figura C.1, em anexo.

Os componentes ligados ao *Arduino* são: módulo *XBee*, os dois *encoders* acoplados a cada uma das rodas, os dois sensores de distância (ultra-sons e infravermelhos) e bússola digital.

Para a utilização do módulo *XBee* e das comunicações sem fios que este possibilita, basta conectar as suas portas $D_{OUT}$ e $D_{IN}$ aos pinos Rx (D0) e Tx (D1) do *Arduino*, além de lhe fornecer a alimentação necessária.

Os *encoders*, tal como foi explicado no parágrafo **3.3.5**, produzem dois sinais de saída. Para fazer a sua leitura com um *trigger* externo, estes sinais são ligados a portas de *interrupt* do *Arduino*. No entanto, como o *Arduino Duemilanove* (bem como o *Uno*) possui apenas duas entradas deste tipo e neste projecto utilizam-se dois *encoders*, apenas um sinal de saída de cada *encoder* é ligado a uma entrada *interrupt* (a outra saída é conectada a qualquer outra porta digital "normal" disponível). Este facto tem como consequência a redução da resolução máxima para metade, ou seja, para 24 contagens por revolução. O *encoder* esquerdo tem as suas saídas OUT_A e OUT_B ligadas aos pinos D2 e D11, respectivamente. Analogamente, o *encoder* direito tem as suas respectivas saídas conectadas às portas D3 e D10. Ambos os *encoders* são alimentados nas portas VCC e GND.

O sensor de distância IR é o mais simples em termos de ligações. Este gera um sinal analógico de tensão à sua saída VOUT que é conectado a uma entrada analógica do *Arduino* (A0, neste caso).

Tanto o sensor de distância de ultra-sons como a bússola digital têm os seus sinais SCL e SDA ligados às portas A5 e A4 do *Arduino*, respectivamente. Isto deve-se ao facto de a comunicação com estes dispositivos ser feita conforme a norma I2C, o que explica também a utilização das duas resistências de "*pull-up*" de 1K8Ω.

O *shield* de *drive* de motores tem três elementos ligados a si: dois motores DC e um servomotor.

Para controlo dos motores das rodas, são utilizadas as saídas do *shield* associadas aos motores M3 e M4. Não são utilizados os motores M1 e M2 porque as respectivas portas estão ligadas à porta D2 do *Arduino*, que é uma entrada de *interrupt* necessária para outro propósito (*encoders*). O motor M3 está associado à roda direita e o motor M4 à roda esquerda. A polaridade da ligação do motor influencia o seu sentido de rotação. Caso o sentido não esteja em conformidade com o que está programado no *Arduino*, basta trocar a polaridade da ligação.

Por fim, o servomotor utilizado tem apenas três pinos: dois para alimentação (V+ e GND) e um para controlo (SIG). Estes são conectados ao *jumper* S2 do *shield* de motores, que redirecciona as três ligações directamente para o *Arduino*, incluindo a alimentação; o sinal SIG é ligado ao pino D9 do *Arduino*, que gera uma saída digital PWM para controlo do servomotor.

Note-se que os elementos não são todos alimentados pela mesma fonte de alimentação. No esquema eléctrico estão representadas as duas alimentações (constituídas por pilhas) independentes para *Arduino* e *shield* de motores, representadas por VCC1 e VCC2, respectivamente. A alimentação do *Arduino* consiste em 8 pilhas em série; o alimentação do shield consiste em 8 pilhas, paralelas quatro a quatro.

O veículo robótico completo, montado seguindo as especificações do esquema eléctrico apresentado acima, está apresentado na **Figura 5.1**, abaixo.

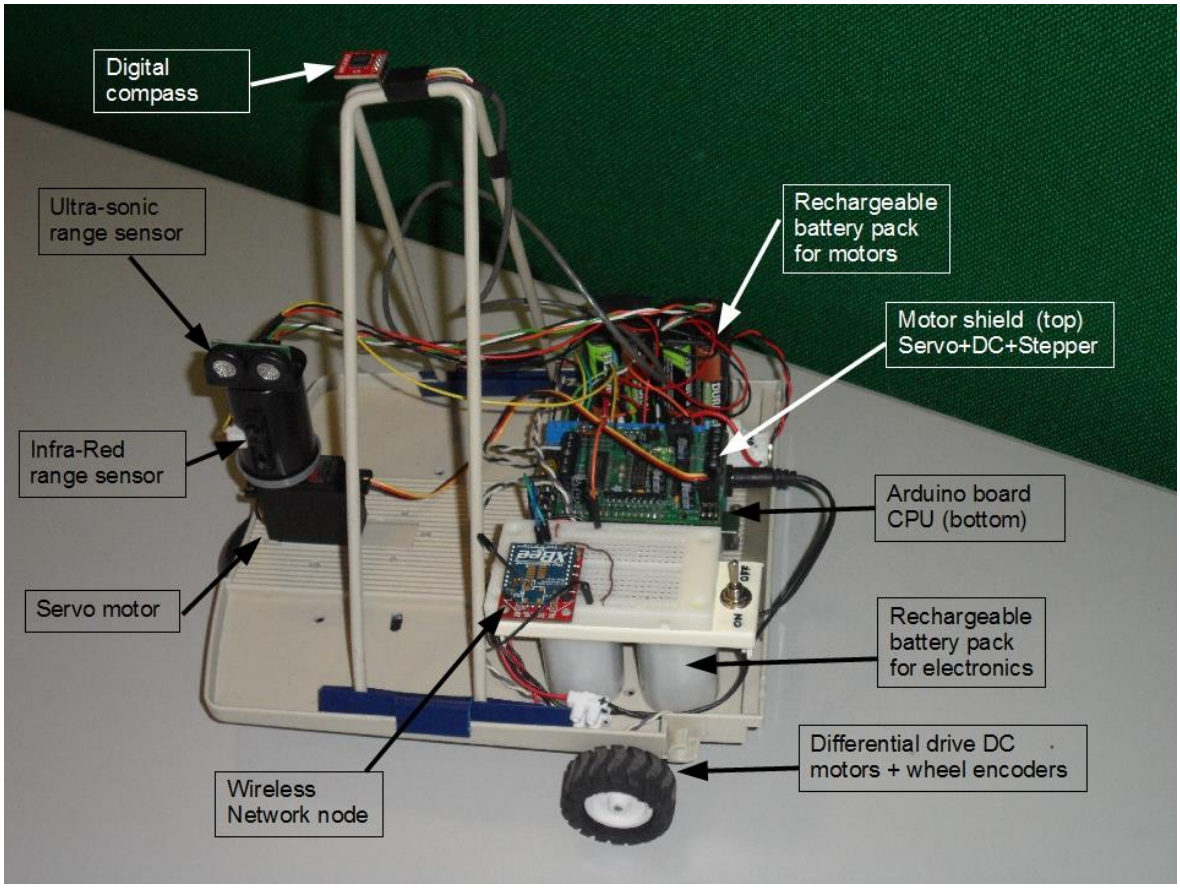



**Figura 5.1** – Estado final de construção do veículo robótico ***GEO-Rover.***

Na **Figura 5.2** estão ilustradas algumas características da geometria do kit robótico, nomeadamente a distância de separação das rodas e a distância entre o eixo das rodas e a localização dos sensores de distância. Esta é relevante para o algoritmo de navegação, para calcular a localização dos sensores no referencial global através da aplicação de uma matriz de rotação em 2D.

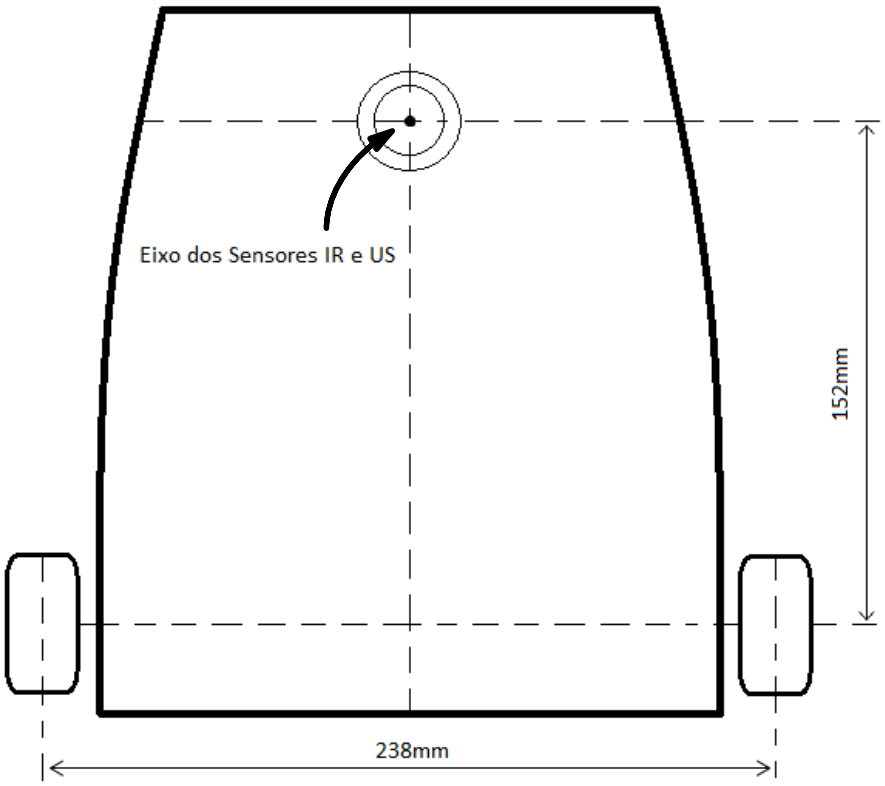



**Figura 5.2** – Esquema com dimensões do kit robótico.

## 5.4. Odometria

Quando se pretende implementar um Modelo Estocástico para o movimento de um veículo robótico, na verdade pretende-se um algoritmo que implemente a seguinte expressão matemática:

$$p(x_t|u_t, x_{t-1}) \qquad 5.1$$

A expressão acima representa a probabilidade de o *robot* estar na posição $x_t$, assumindo que no instante anterior $(t-1)$ se encontrava no estado $x_{t-1}$ e lhe foi dado o comando para se deslocar segundo a instrução $u_t$. Note-se que a variável $x_t$ representa uma possível pose do *robot* no instante $t$, e $u_t$ é tipicamente uma translação ou uma rotação.

Existem dois Modelos que descrevem satisfatoriamente o comando $u_t$: o Modelo de Movimento com base na Velocidade, e o Modelo de Movimento com base nos Deslocamentos **[30]**. Neste trabalho é adoptada uma versão simplificada do segundo modelo. Este assume que os deslocamentos do *robot* (indicados pela odometria) correspondem a comandos de deslocamento e rotação em 2D simplificados no formato $[\Delta x, \Delta y, \Delta\theta]$. Embora este assunto esteja fora do âmbito deste trabalho, salienta-se que este modelo simplificado só é válido se a odometria for obtida com uma frequência relativamente elevada, como é no caso presente; caso contrário, deverá ser aplicado o Modelo de Deslocamentos tal como descrito em **[30]**.

A odometria indica o deslocamento de cada roda. Com as seguintes expressões:

$$D = \frac{D_L + D_R}{2} \qquad 5.2$$

$$\theta = \frac{D_L - D_R}{d} \qquad 5.3$$

conseguem-se traduzir os deslocamentos $D_L$ e $D_R$ das rodas (esquerda e direita, respectivamente) para uma translação $D$ e uma rotação $\theta$ (em radianos) no trajecto do veículo robótico, em que $d$ é o comprimento do eixo das rodas **[41]**. A **Figura 5.3** ilustra a cinemática do veículo, que justifica as expressões **5.2** e **5.3**.

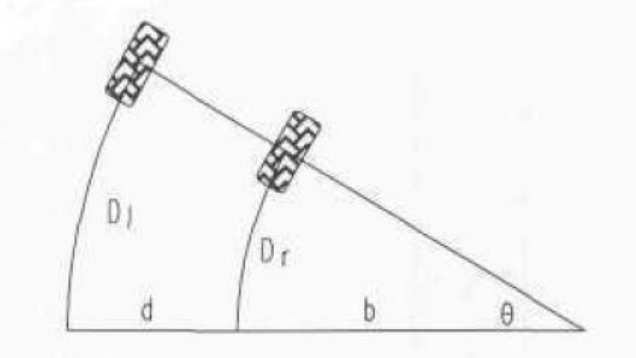



**Figura 5.3** – Esquema base da cinemática que permite deduzir as equações **5.2** e **5.3** **[41]**.

O modelo de movimento aplica sucessivamente o algoritmo (que descreve a expressão **5.1**) às translações e rotações. Como os valores de odometria provenientes dos *encoders* são portadores de erro, este vai sendo integrado e acumulado ao longo do tempo e a incerteza associada às estimativas da localização do *robot* vai aumentando, o que no caso dos filtros de partículas se traduz numa crescente dispersão das partículas que representam as hipotéticas posições do veículo, como se pode ver na **Figura 5.4**. Para reduzir este problema, outros dados sensoriais são tidos em conta, tais como medidas de distância. Este assunto é abordado no Capítulo **6**, em que é executado o algoritmo de navegação.

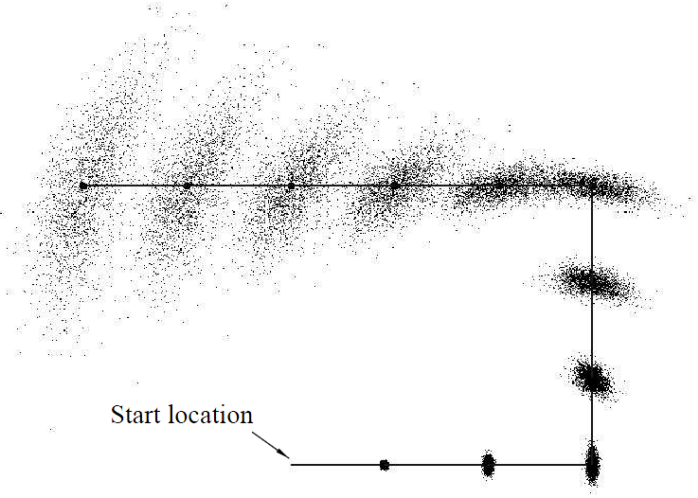



**Figura 5.4** – Influência dos erros da odometria no Modelo de Movimento do veículo robótico **[30]**.

## 5.5. Integração do controlo de motores (seguimento de trajectórias)

Apresenta-se agora o controlo de motores no seguimento de uma trajectória. Aqui é usada a odometria para o controlo da trajectória. Poderia ser usada também para localização global do *robot*, mas neste ponto pretende-se analisar o controlo apenas.

Os comportamentos implementados no veículo robótico em que são aplicados métodos de controlo de motores, são os STRAIGHT e FOLLOWALL. Nestes comportamentos, pretende-se manter determinados parâmetros constantes ao longo da trajectória: direcção frontal no primeiro caso e distância à parede, no segundo. Para o conseguir, é necessário integrar no sistema um método eficaz de controlo dos motores. Tal como foi já referido, o controlo em ambos os cenários é feito através de um algoritmo PID devidamente afinado para cada um dos casos.

No caso do comportamento STRAIGHT, como o objectivo é seguir em frente segundo uma trajectória rectilínea, pretende-se que ambas as rodas se desloquem à mesma velocidade. Neste processo, os elementos envolvidos são: os motores DC (rodas) e os *encoders* acoplados a cada uma das rodas. As entradas do PID (implementado no *Arduino*) são as leituras dos *encoders*, e as saídas são as velocidades a aplicar a cada um dos motores DC, por forma a contrariar eventuais diferenças entre os deslocamentos de cada roda.

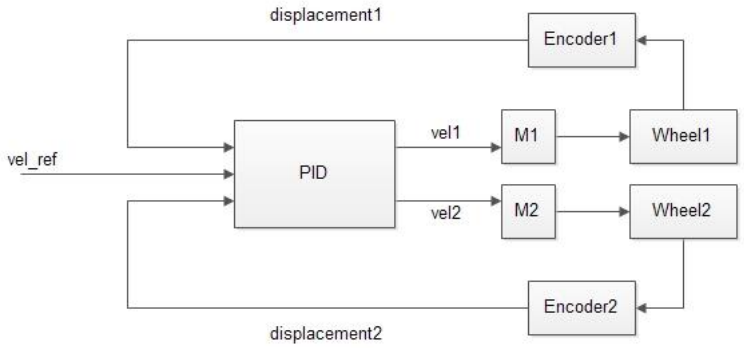



**Figura 5.5** – Diagrama de blocos do controlo de motores numa trajectória rectilínea.

No comportamento FOLLOWALL, o objectivo é manter uma distância constante à parede lateral. Portanto é necessário medir essa distância e controlar os motores das rodas para contrariar os desvios. Os elementos relevantes aqui são: o(s) sensor(es) de distância à parede e os motores DC (rodas). A entrada do sistema de controlo PID (ou entradas, caso se utilizem leituras de mais de um sensor) é a medição da distância à parede obtida pelo sensor IR, e as saídas são as velocidades individuais a aplicar a cada um dos motores DC.

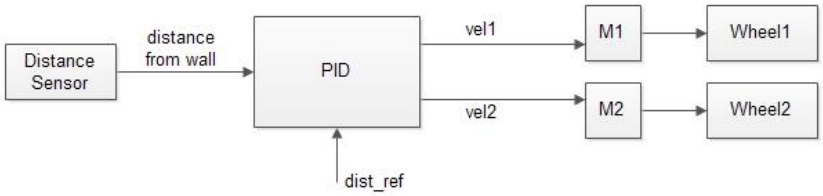



**Figura 5.6** – Diagrama de blocos do controlo de motores no seguimento de uma parede.

### 5.5.1. Trajectória rectilínea

Com os sinais provenientes dos *encoders* acoplados às rodas e utilizando os *interrupts* do *Arduino*, são registados os deslocamentos de cada roda. Periodicamente (sensivelmente de 100ms em 100ms), os valores actuais dos contadores associados a cada roda são guardados (para serem utilizados pelo PID) e repostos a zero novamente (não só torna os cálculos mais simples, mas também evita o problema de *overflow*). No PID, é calculado o erro a corrigir como a diferença entre os deslocamentos de cada roda, conforme pode ser descrito pelo algoritmo abaixo.

```
error = displacement_left – displacement_right;
k = calcPID(error);
right_wheel_speed = max_speed + error*k;
left_wheel_speed = max_speed – error*k;
```

Nas figuras seguintes estão representados os resultados mais relevantes provenientes da actuação do comportamento STRAIGHT. Todos os dados estão com uma taxa de amostragem de 10Hz. Não são apresentados os valores de cada *encoder* em cada instante, mas são representadas outras informações mais relevantes daí retiradas, como o deslocamento e a orientação. São ainda representados os valores de cada componente do PID em cada instante.

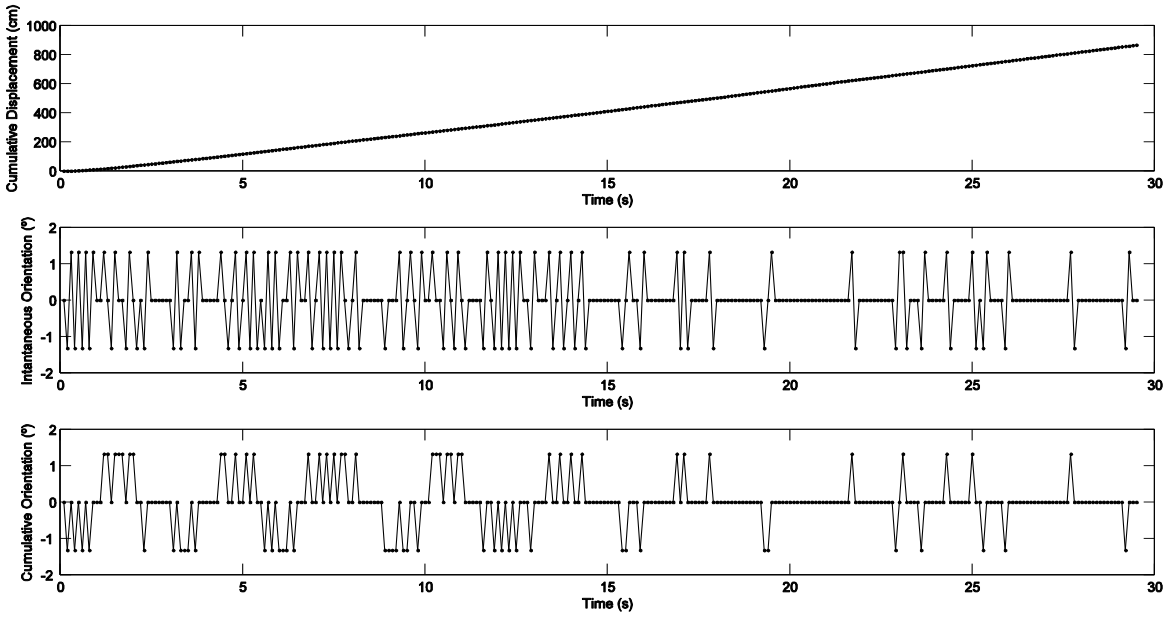



**Figura 5.7** – Deslocamento, orientação instantânea e orientação acumulada do *robot* durante uma trajectória rectilínea.

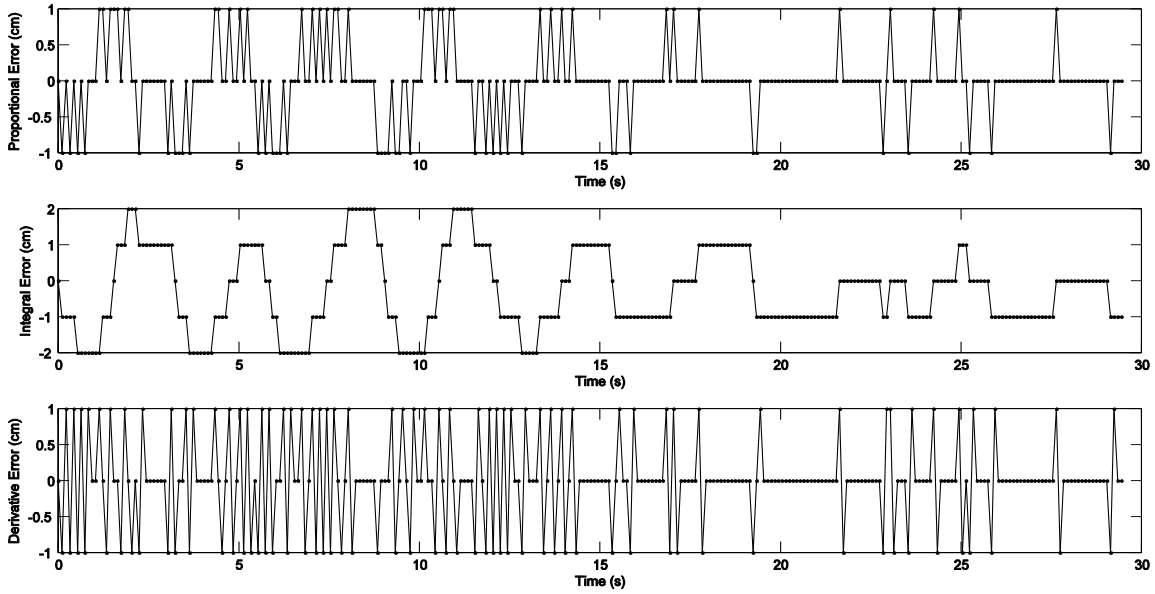



**Figura 5.8** – Erros de cada componente do PID durante o controlo numa trajectória rectilínea.

Como se pode verificar pelos gráficos da **Figura 5.7**, a orientação instantânea vai sofrendo alguns desvios aproximadamente simétricos ao longo do tempo. Isto significa que cada vez que o veículo tem um desvio para um dos lados, tenta seguidamente contrariá-lo, curvando para o lado contrário. Isto conclui-se mais facilmente na orientação acumulada, pois observa-se que em termos médios o veículo mantém uma orientação relativa de 0°, ou seja, sem desvios. No gráfico superior da figura em causa tem-se o deslocamento ao longo do tempo. Daqui observa-se uma ligeira aceleração inicial, mantendo depois a velocidade constante ao longo de todo o percurso de cerca de 8 metros.

Na **Figura 5.8** encontram-se os gráficos dos componentes do PID. Observa-se a acção de cada um, dado que corrigem aspectos diferentes do sistema. O proporcional é reduzido porque a orientação sofre pequenos desvios de cada vez. O derivativo também tem erros reduzidos porque o veículo sofre desvios que não são muito "bruscos". O integral é o componente que apresenta mais erro, pois representa o erro acumulado do deslocamento diferencial ao longo do tempo.

## 5.5.2. Seguimento de parede

No seguimento de uma parede/superfície, o princípio de controlo consiste em medir a distância actual do *robot* à superfície, calcular o erro comparando-a com a distância de referência à parede, e actuar nos motores por forma a corrigir eventuais desvios. A correcção da distância actual à parede é feita periodicamente a cada 100ms.

O sensor de distância IR faz medições a uma frequência de 20Hz, sendo calculada a média de cada duas medidas para fornecer ao PID uma distância à parede menos sensível a erros do sensor. Desta forma, a taxa de amostragem usada pelo PID é de 10Hz. As medidas de distância à parede são comparadas com a distância de referência (tipicamente 20cm), sendo a diferença entre elas o erro a ser corrigido. Este processo está representado no algoritmo abaixo.

```
error = distance_ref – distance_sensor_read;
k = calcPID(error);
right_wheel_speed = max_speed – error*k;
left  wheel  speed = max  speed + error*k;
```

Nas figuras seguintes estão representados os dados mais importantes resultantes do seguimento de uma parede recta (sem portas). A taxa de amostragem em todos casos é de 10Hz, e podem observar-se as distâncias à parede, os erros associados a cada componente do PID e outras informações extraídas dos valores de cada *encoder*.

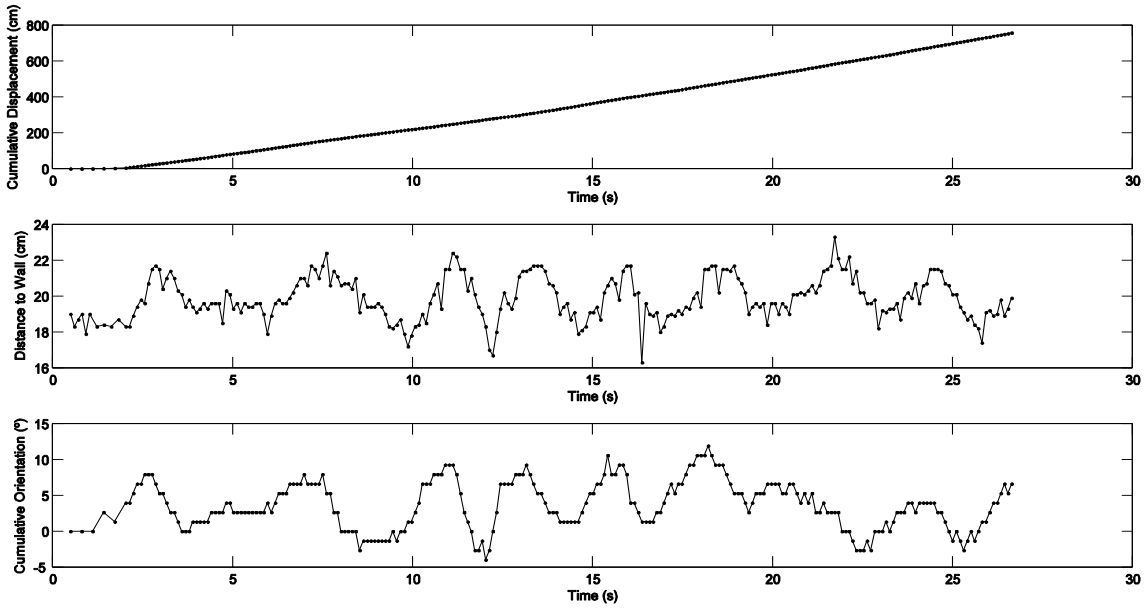



**Figura 5.9** – Deslocamento, distância à parede e orientação acumulada do *robot* durante o seguimento de uma parede.

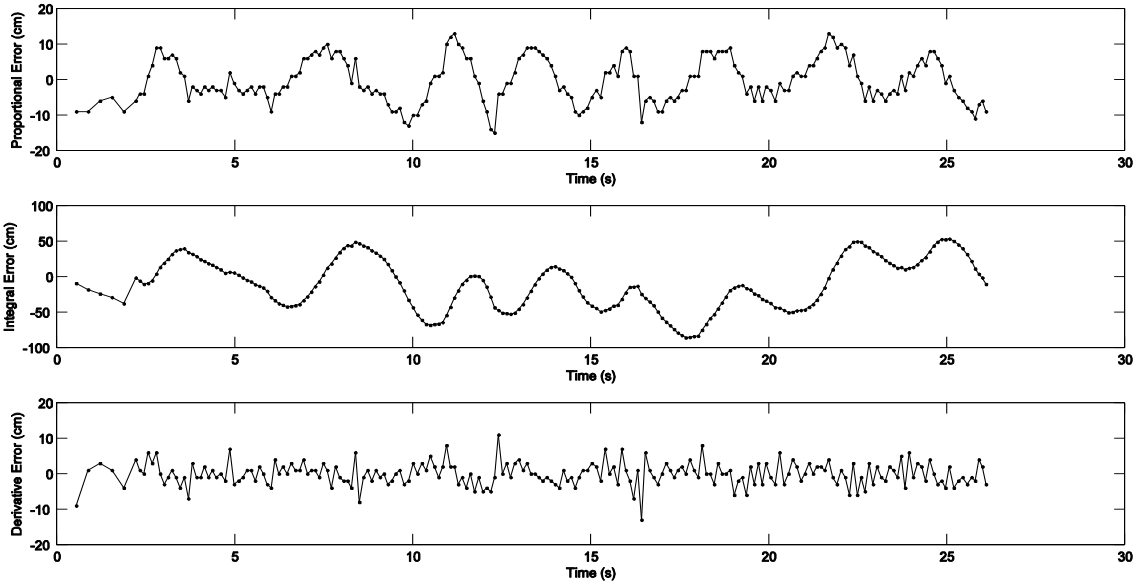



**Figura 5.10** - Erros de cada componente do PID durante o controlo no seguimento de uma parede.

Na **Figura 5.9** estão apresentados os resultados relacionados com o movimento do *robot*. No gráfico superior está traçado o deslocamento do veículo ao longo do tempo.

Observa-se uma ligeira aceleração inicial, movendo-se a velocidade constante no resto do percurso de cerca de 8 metros. No gráfico da distância à parede, verifica-se uma oscilação em torno dos 20cm, que é a distância que o controlo tenta manter à parede. Esta oscilação tem uma amplitude de aproximadamente 4cm e é causada em parte por erros nas medições de distância à parede e em parte também por limitações do algoritmo de controlo. No gráfico inferior da mesma figura é apresentada a orientação do veículo ao longo do trajecto. Esta está em concordância com o gráfico da distância à parede, pois cada desvio em relação à parede traduz-se na realidade numa alteração de orientação do veículo, isto é, na sua direcção de deslocamento.

Na **Figura 5.10** encontram-se os gráficos com os erros associados a cada componente do PID. O proporcional representa o erro na distância à parede (quando comparado com os 20cm de referência) em cada instante. O integral representa o erro acumulado ao longo do tempo, e aqui é mais visível o movimento ondulatório do *robot*. O derivativo representa a variação do erro instantâneo. Aqui são visíveis os desvios sistemáticos na trajectória (apesar de reduzidos em amplitude), e devem-se aos constantes ajustes na trajectória do veículo.

Comprova-se a eficácia do controlo com PID no seguimento de uma parede.

## 5.5.3. Controlo noutras trajectórias

### 5.5.3.1. Trajectórias não rectilíneas

Foram ainda testados outros exemplos de seguimentos de trajectórias, mas que não foram mais do que variações de uma das trajectórias já apresentadas.

Colocou-se o veículo robótico, executando o comportamento de seguimento de paredes, paralelamente a três tipos de paredes diferentes (ver Capítulo **4.2.1** e suas subsecções, nomeadamente a **Figura 4.6**): convexa (em torno de um pilar redondo), côncava (usando cartão para criar uma superfície redonda), e uma mistura de ambos. Em todos os casos o *robot* cumpriu o objectivo e seguiu a superfície lateral, independentemente da sua forma. Porém, quando o raio do arco de circunferência da superfície que o *robot* segue se torna muito reduzido, torna-se difícil ou mesmo impossível de seguir. Esta situação é mais notória nas superfícies côncavas. Uma pequena melhoria que se mostrou eficaz para esta situação é direccionar o sensor de distância ligeiramente para a frente, em vez de estar apontado exactamente perpendicular à parede.

### 5.5.3.2. Trajectória com curvatura constante

A partir do comportamento de seguir uma trajectória rectilínea, obteve-se a trajectória curvilínea. Na trajectória rectilínea faz-se o controlo para manter nula a diferença entre as velocidades das rodas; na curvilínea o objectivo é manter uma diferença de velocidades entre as duas rodas. Assim, o veículo robótico descreve um raio de curvatura aproximadamente constante, dependente dessa diferença.

Porém, este comportamento não foi alvo de estudo, pois pressupôs-se que o veículo robótico se deslocava em linhas rectas, sendo estas ligadas com rotações de 90°.

## 5.6. Exemplo Prático do Controlo das Trajectórias

Por forma a avaliar o desempenho do controlo dos motores no seguimento de trajectórias, efectuaram-se testes com o veículo robótico num circuito bem definido. Nestes testes, o objectivo foi colocar o *robot* a seguir uma parede ao longo do corredor e voltar ao início seguindo a parede oposta. O trajecto no corredor está representado na **Figura 5.11**.

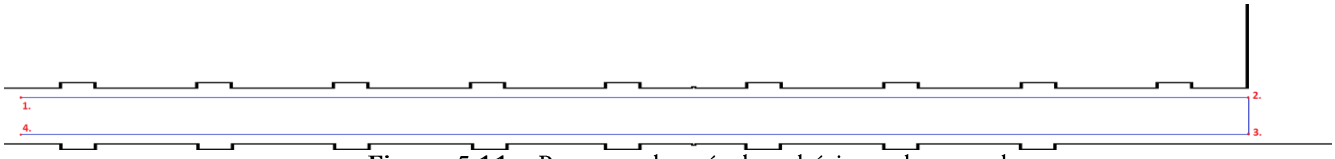



**Figura 5.11** – Percurso do veículo robótico pelo corredor.

No decorrer do teste, as transições de estado necessárias para o cumprimento do percurso são feitas através de comandos enviados a partir do PC. O teste consiste maioritariamente no seguimento da parede segundo o comportamento FOLLOWALL (entre os pontos 1 e 2, e 3 e 4), tendo apenas duas rotações de 90° (pontos 2 e 3), e um pequeno troço de deslocação em frente segundo o comportamento STRAIGHT (entre os pontos 2 e 3).

Em todas as deslocações do veículo, este faz a aquisição sensorial correspondente aos comportamentos que vai tomando. Nomeadamente no FOLLOWALL, o sensor IR aponta para a parede do corredor mais próxima enquanto o sonar está direccionado para a parede oposta. Estes dados podem ser usados para Navegação em tempo real, para localização *offline* ou para análise.

No mesmo corredor, foram feitos os testes em dois cenários diferentes, mantendo o trajecto do veículo conforme a **Figura 5.11**. O **Cenário 1** corresponde ao corredor tal como é na realidade e está representado na Figura A.1. Para o **Cenário 2**, ocultaram-se duas portas do corredor com cartão, como está ilustrado na Figura A.2.

Ainda em cada cenário, foram feitos dois testes, que correspondem às duas abordagens à implementação do controlo dos motores explicadas anteriormente na subsecção **4.2.1.5** .

### 5.6.1. Cenário 1

Na **Primeira Abordagem** ao controlo dos motores, a verificação e eventual ajuste do *set_point* é efectuada apenas quando há detecções de reentrâncias das portas. Na **Segunda Abordagem**, esta verificação do *set_point* é efectuada em todos os instantes. Para mais detalhe, as duas abordagens ao controlo dos motores estão descritas na subsecção **4.2.1.5.**

Os resultados destas duas abordagens de controlo são analisados a seguir.

#### 5.6.1.1. Primeira Abordagem de controlo

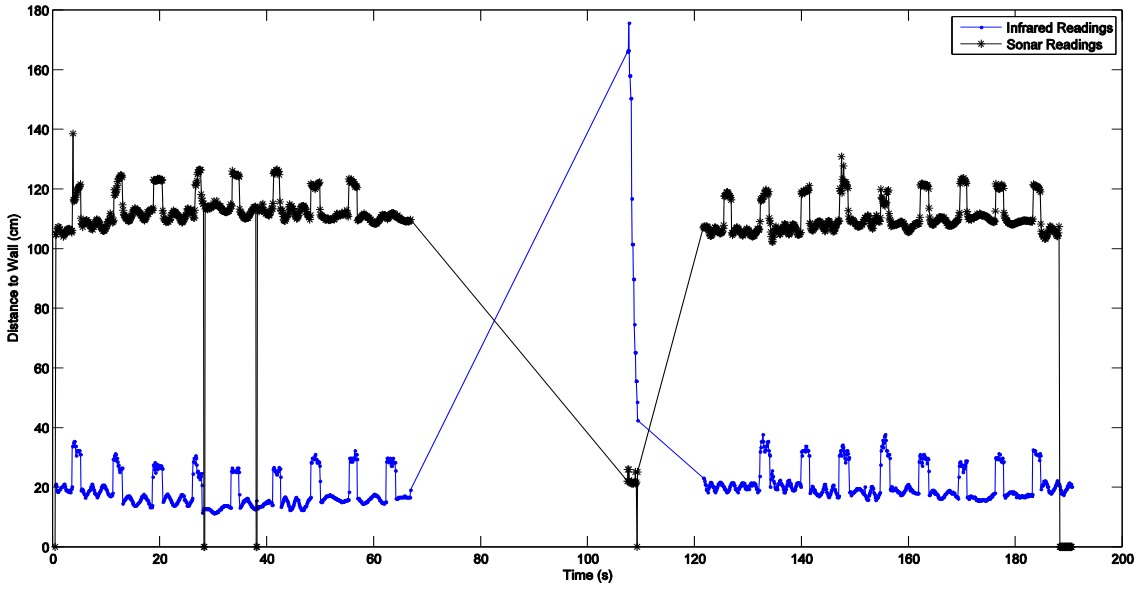



**Figura 5.12** – Dados adquiridos pelos sensores de distância ao longo do percurso na **Primeira Abordagem** no **Cenário 1.** Os dados representados perto dos 110s correspondem ao momento depois de o *robot* ter rodado 90° e se encontra a atravessar o corredor (trajecto entre 2 e 3 da **Figura 5.11**), com o sensor IR direccionado para a parede.

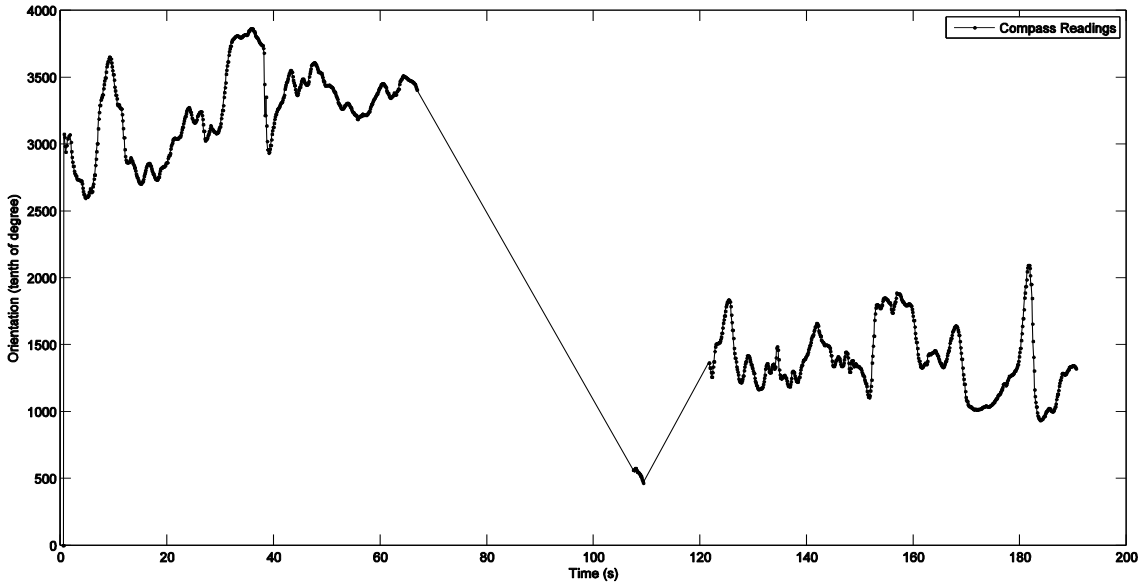



**Figura 5.13** – Dados adquiridos pela bússola ao longo do percurso na **Primeira Abordagem** no **Cenário 1.**

No gráfico da **Figura 5.12** estão representados os dados recolhidos pelo sensor infravermelhos e pelo ultra-sons ao longo do tempo. Estes dados estão em centímetros e representam as distâncias medidas por cada sensor a um obstáculo, neste caso a parede.

Podem identificar-se claramente três fases distintas nos dados, principalmente com a ajuda do sensor infravermelhos:

- Uma primeira fase em que o veículo vai detectando as paredes e as respectivas portas (corresponde ao trajecto entre os pontos 1 e 2 na **Figura 5.11**);
- Uma segunda em que os dados do infravermelhos são basicamente uma recta com declive negativo, ou seja, o *robot* aproxima-se de uma parede (trajecto entre 2 e 3 da **Figura 5.11**, em que o robot vai em direcção à parede de baixo, com o IR apontado em frente);
- Por fim uma terceira fase em que se voltam a identificar as portas no corredor, que corresponde ao trajecto entre 3 e 4 da **Figura 5.11**, em que os sensores de distância apontam novamente perpendicularmente à direcção de deslocamento do veículo.

É também de salientar a concordância que existe entre os dois sensores, que detectam as saliências sempre simultaneamente. Isto faz sentido, pois o corredor é simétrico e tem quase todas as portas à mesma distância, exceptuando um excerto mais comprido a meio do corredor (ver Figura A.1).

Apesar de os resultados serem bastante satisfatórios, notam-se dois tipos de oscilação no trajecto do veículo robótico. A oscilação de pequena amplitude e frequência mais elevada deve-se às frequentes correcções na trajectória do *robot* impostas pelo PID, enquanto as oscilações de maior amplitude e mais espaçadas se devem ao facto de o *set_point* ser corrigido a intervalos de tempo maiores. No entanto, o veículo consegue sempre seguir a parede e continua o seu trajecto enviando periodicamente os dados sensoriais.

Nos dados adquiridos pela bússola, as três fases do percurso da **Figura 5.11** são mais facilmente identificadas. Os pontos 2 e 3 correspondem às variações bruscas na orientação da bússola: instantes 105s e 120s da **Figura 5.13**.

Nota-se ainda que, apesar de o *robot* seguir uma trajectória aproximadamente rectilínea paralela à parede, os dados obtidos pela bússola variam consideravelmente (desvio superior a 100°). Isto deve-se não só às correcções sistemáticas na trajectória do veículo, mas principalmente às anomalias magnéticas existentes no ambiente. Observe-se, por exemplo, a anomalia detectada entre os instantes 30s e 40s, em que o veículo passou por uma caixa de incêndio instalada na parede. Devido ao seu material predominantemente metálico, esta causa uma anomalia magnética que se observa claramente nos dados da bússola. Este é apenas um exemplo concreto, pois ao longo do tempo é visível a existência de outras perturbações magnéticas.

Nos três intervalos de tempo de aquisição da bússola, é notório que as anomalias sucedem em volta de orientações distintas. Isto reflecte as três etapas do percurso do *robot*.

### 5.6.1.2. Segunda Abordagem de controlo

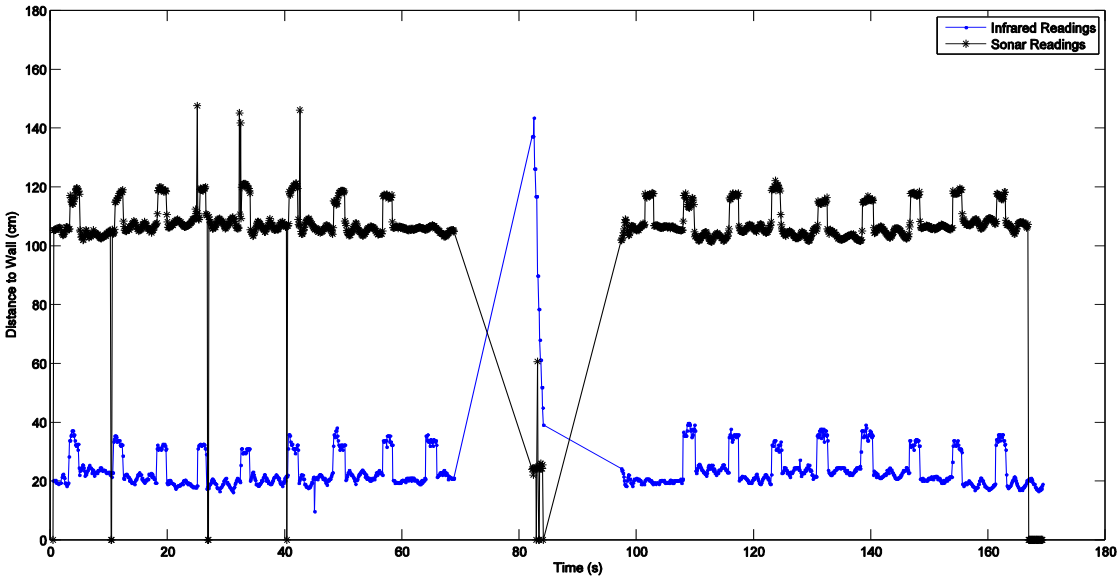



**Figura 5.14** – Dados adquiridos pelos sensores de distância ao longo do percurso na **Segunda Abordagem** no **Cenário 1.**

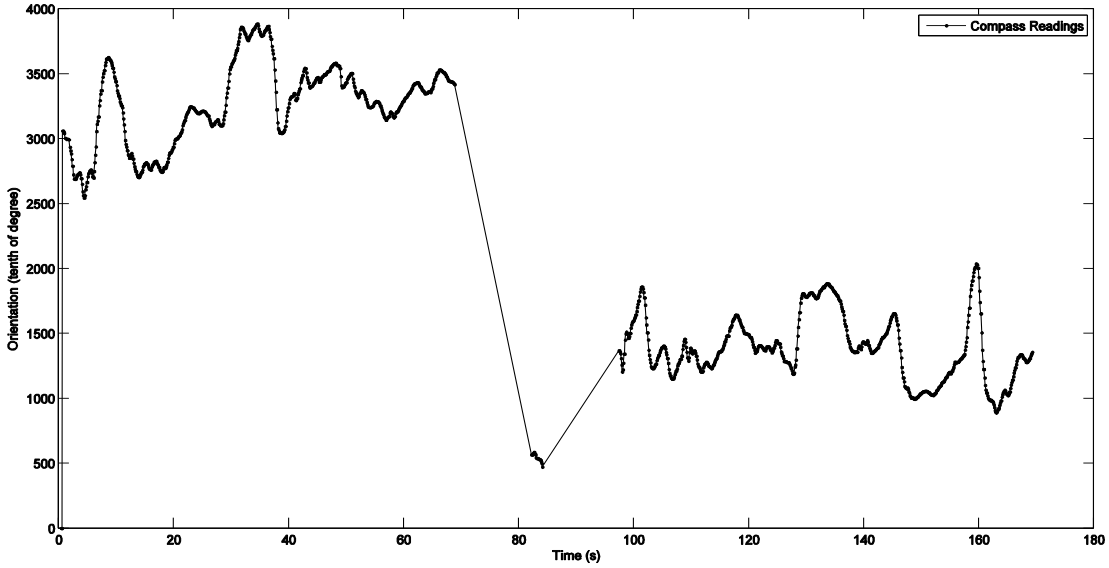



**Figura 5.15** – Dados adquiridos pela bússola ao longo do percurso na **Segunda Abordagem** no **Cenário 1.**

Tal como na abordagem anterior, também aqui são facilmente identificadas as três fases do percurso bem como as portas do corredor. A principal diferença é, de facto, a que está relacionada com o tipo de abordagem no controlo dos motores. Como aqui o *set_point* é corrigido sistematicamente, a longo prazo o veículo robótico desvia-se menos dos 20cm assumidos como a distância referência da parede. Compare-se, por exemplo, o intervalo de tempo [0s; 40s] da **Figura 5.12** e **Figura 5.14.** No primeiro caso é notória a maior aproximação do *robot* à parede (observar os dados do sensor IR).

Os dados recolhidos pela bússola são consistentes com os obtidos na primeira abordagem. As discrepâncias observadas explicam-se por eventuais desvios no trajecto relativamente ao teste anterior, pois se num dado ponto onde existe uma anomalia magnética o veículo se aproximar mais da parede do que noutro teste, isso reflecte-se nos dados sensoriais adquiridos pela bússola.

### 5.6.2. Cenário 2

As abordagens ao controlo dos motores usadas neste cenário são as mesmas que no primeiro. A única alteração é de facto a ocultação de duas portas, tal como pode ser observado na Figura A.2.

#### 5.6.2.1. Primeira Abordagem de controlo

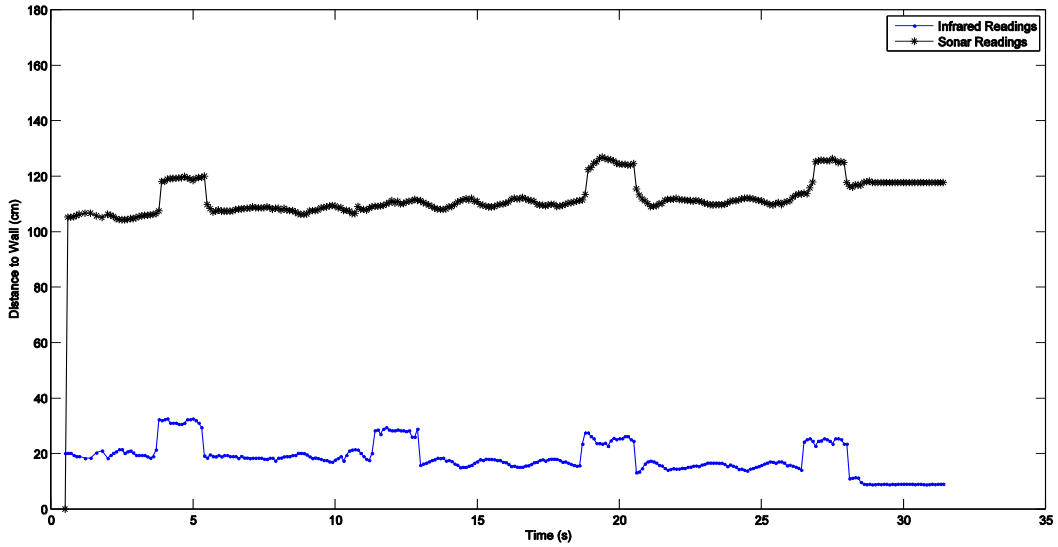



**Figura 5.16** – Dados adquiridos pelos sensores de distância ao longo do percurso na **Primeira Abordagem** no **Cenário 2.**

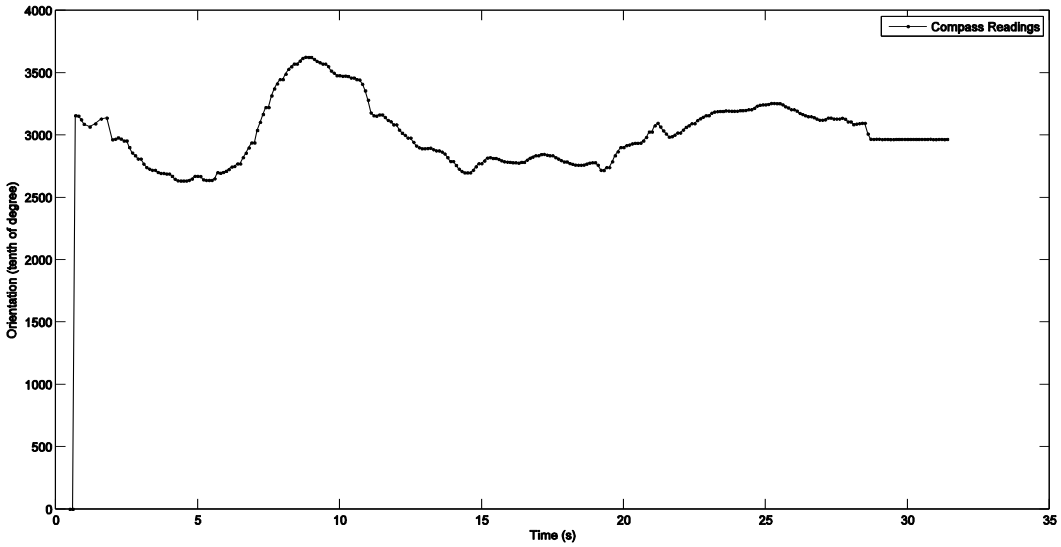



**Figura 5.17** – Dados adquiridos pela bússola ao longo do percurso na **Primeira Abordagem** no **Cenário 2.**

Como se pode ver pelos gráficos, os dados dos sensores de distância estão consistentes com o que acontecia na **Primeira Abordagem** no **Cenário 1** (**Figura 5.12** e **Figura 5.13**). A única alteração relevante neste cenário é a inexistência de duas portas, de acordo com o mapa da Figura A.2.

Relativamente aos dados da bússola, estes são muito semelhantes aos obtidos nos testes precedentes. Isto deve-se ao facto de o percurso do *robot* ser o mesmo e as alterações ao cenário serem feitas com material que não afecta o campo magnético do meio envolvente.

### 5.6.2.2. Segunda Abordagem de controlo

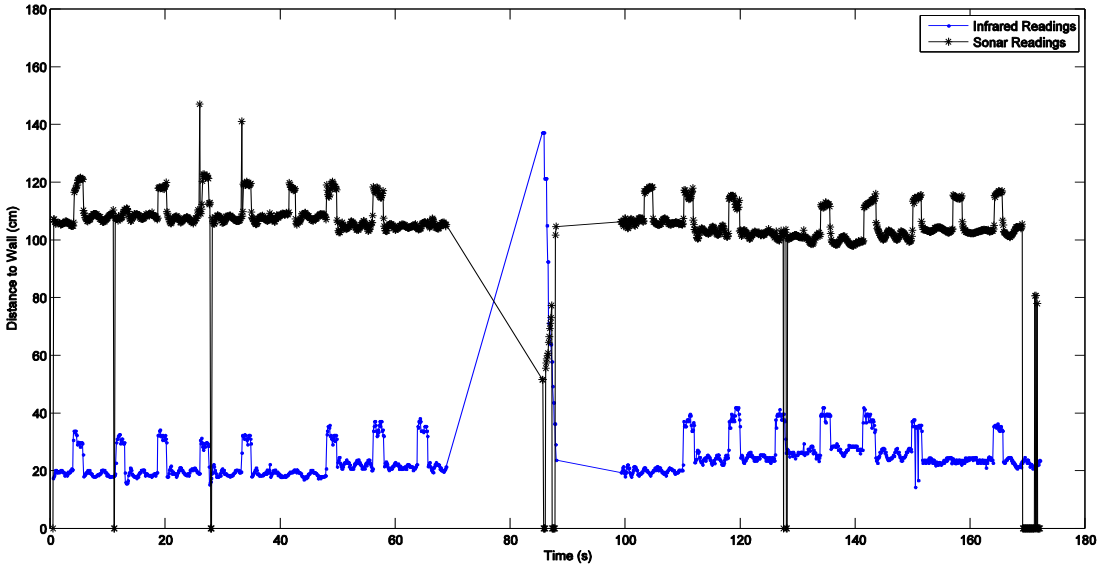



**Figura 5.18** – Dados adquiridos pelos sensores de distância ao longo do percurso na **Segunda Abordagem** no **Cenário 2.**

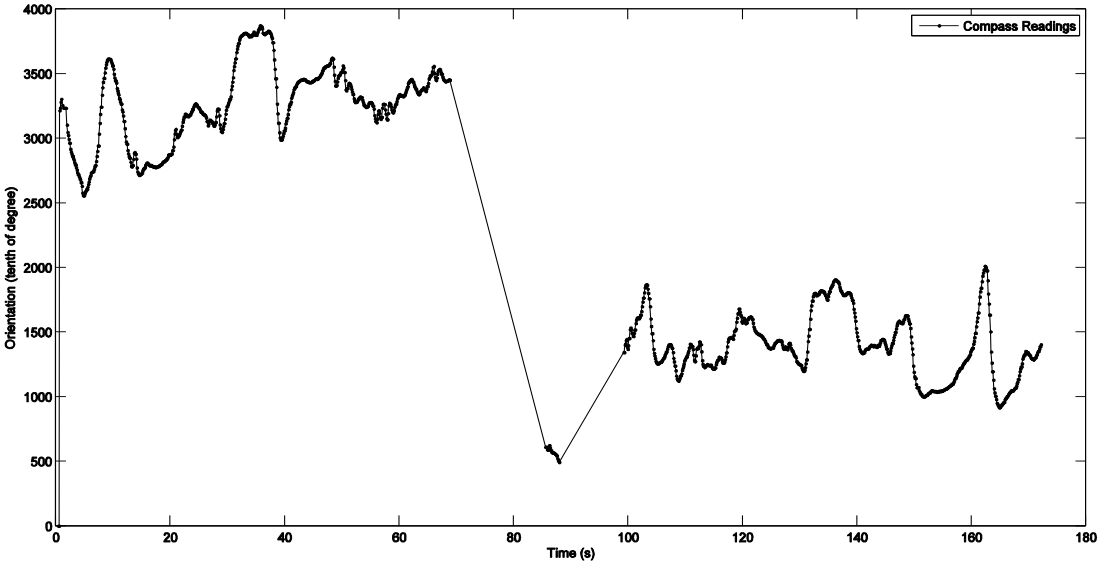



**Figura 5.19** – Dados adquiridos pela bússola ao longo do percurso na **Segunda Abordagem** no **Cenário 2.**

Aqui as conclusões são semelhantes às que foram tiradas na **Segunda Abordagem** do **Cenário 1** (**Figura 5.14** e **Figura 5.15**). Pela **Figura 5.18** conclui-se que a longo prazo o veículo robótico oscila menos devido à abordagem utilizada no controlo dos motores, e que os dados dos sensores representam correctamente o ambiente observado (Figura A.2) à medida que o *robot* vai avançando. No intervalo de tempo [110s; 150s] constata-se um ligeiro afastamento da parede. Isto deve-se ao facto da verificação do *set_point* servir principalmente como mecanismo de segurança para evitar que o *robot* se aproxime demasiado da parede, não fazendo qualquer correcção caso este se afaste dela.

Os dados da bússola são idênticos aos dos testes precedentes. As condições envolventes são as mesmas, bem como o percurso do *robot*, portanto os dados adquiridos devem ser aproximadamente os mesmos.

Conclui-se que em ambas as abordagens e em ambos os cenários o controlo da trajectória é bem conseguido, apresentando cada uma das abordagens vantagens e desvantagens.

## 5.7. Integração da Comunicação Sem Fios

A comunicação entre o veículo robótico e o PC é uma componente muito importante do trabalho, embora não seja necessária permanentemente. Por exemplo quando o *robot* está a fazer uma rotação, o importante é que ele fique com a orientação final desejada, não sendo tão relevante que ele detecte ou não objectos em seu redor. Neste caso, o controlo é feito localmente no *robot*, sem necessidade de comunicação com o PC. Já no caso em que a tarefa actual do *robot* é fazer um varrimento sensorial, o mais importante é que todos os dados sensoriais sejam enviados para o PC, para serem processados.

Os comportamentos em que existe comunicação entre PC e *robot* são: IDLE, STRAIGHT, SCANNING e FOLLOWALL. No estado IDLE, o veículo robótico está estacionário à espera de receber um comando proveniente do PC, pelo que a comunicação é apenas no sentido do PC para o *robot*. Nos comportamentos STRAIGHT e FOLLOWALL o veículo robótico vai recolhendo e enviando dados sensoriais para o PC enquanto executa a tarefa correspondente, e vai constantemente verificando se foi recebido algum comando do PC – aqui a comunicação é bidireccional. Finalmente, durante o comportamento SCANNING, o *robot* faz uma recolha sensorial mais detalhada que é enviada para o PC, transitando de estado quando conclui a tarefa. Neste caso, a comunicação faz-se no sentido do *robot* para o PC.

A criação e envio dos pacotes de dados, bem como a recepção, é feita conforme foi apresentado no Capítulo **2**, recorrendo aos módulos *XBee*. A interpretação dos tipos de dados, quer no PC quer no *Arduino*, segue a estrutura apresentada na secção **4.5.1.**

Os elementos físicos do *robot* que estão associados às trocas de dados são: módulos *XBee* (no veículo robótico e no PC) e o *Arduino*. Estes componentes e as ligações necessárias estão ilustrados no esquema eléctrico da **Figura 5.20,** abaixo.

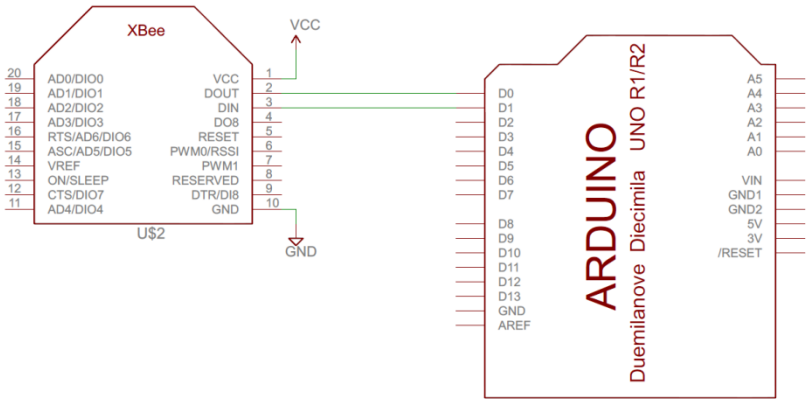



**Figura 5.20** – Ligações entre *Arduino* e módulo *XBee*.

## 5.8. Representação e Estimação do Estado do Veículo em Tempo Real

O estado do veículo consiste na sua posição no mapa, conjugada com a sua orientação em 2D; pode ainda ser relevante identificar o Estado/Comportamento actual do *robot*. O *software* de navegação na sua versão actual representa a posição estimada do veículo no mapa, sem a sua orientação. No entanto, a navegação utiliza e representa internamente a pose do veículo.

Para poder estimar a posição do veículo robótico, o *software* utiliza um conjunto de dados sensoriais essenciais. Esses dados são: medidas provenientes de um ou mais sensores de distância cujas orientações devem ser devidamente identificadas, e dados de odometria (valores de deslocamentos das rodas do veículo robótico) por forma a saber quanto é que o *robot* avançou ao longo do tempo. Apesar de os dados sensoriais provenientes da bússola digital ainda não estarem integrados no algoritmo de navegação, estes são enviados para o *software* para futura utilização, como por exemplo o mapeamento da assinatura magnética do ambiente onde o veículo robótico efectuou os testes.

No teste realizado com o algoritmo de navegação, existem apenas duas situações em que são enviados dados para o PC: enquanto o *robot* segue a parede, e quando segue em frente em linha recta. Os sensores de distância IR e US são orientados em sentidos opostos. Quando o veículo segue paralelamente à parede, esta encontra-se à sua esquerda a cerca de 20cm, pelo que o sensor mais conveniente para a detectar é o de infravermelhos, pois apesar de ter um alcance mais reduzido, tem maior precisão e maior rapidez de resposta. Assim sendo, o sensor de ultra-sons fica direccionado para a parede oposta do corredor, que se encontra a cerca de 150cm. Quando o *robot* atravessa o corredor, o motor servo altera a orientação dos sensores, ficando o sensor IR apontado para a frente do veículo, o que lhe permite localizar-se em relação à parede.

Tal como foi referido, o algoritmo de navegação estima a posição do veículo robótico com base num filtro de partículas. Isto significa que o *software* não fornece uma, mas sim um número elevado de poses hipotéticas. Estas múltiplas hipóteses são representadas por uma nuvem de pontos dispostos no mapa. Cada um destes pontos corresponde a uma possível localização do veículo robótico, calculada com base nos dados

de odometria e filtradas com base nas medições de distância aos obstáculos representados no mapa. Quanto mais concentrada for a nuvem de pontos, menor é a incerteza assumida pelo filtro de navegação na estimação da posição do *robot* no mapa. Notar no entanto que a incerteza também é representada pelos pesos atribuídos pelo filtro a cada partícula, não apenas pela sua distribuição espacial.

A **Figura 4.8** da secção **4.3** ilustra duas situações de representação do estado do veículo. À direita trata-se de uma estimativa com maior incerteza, pois a nuvem de pontos está bastante dispersa. À esquerda representa-se uma estimativa com menor incerteza associada, como se pode constatar pela concentração dos pontos da nuvem.

### 5.8.1. Discussão da noção de "Tempo-Real"

Nesta secção aborda-se o problema da representação do estado do veículo robótico em tempo-real. É importante esclarecer o que isto significa neste contexto e quais os cenários em que se trata ou não de tempo-real.

Tempo-real significa que todas as consequências resultantes das interacções entre o veículo robótico e o ambiente circundante são monitorizadas e representadas praticamente ao mesmo tempo, à parte de pequenos atrasos desprezáveis. Na realidade nada é instantâneo, e existem sempre atrasos associados ao envio/recepção dos dados, ou mesmo no seu processamento. No entanto, se estes atrasos forem suficientemente pequenos, são desprezados e considera-se que o processo é tempo-real.

A nível do *Arduino*, onde os dados sensoriais são recolhidos e os motores são comandados, existe, de facto, um controlo em tempo-real: por exemplo, se o *robot* detectar um obstáculo muito próximo quando vai a deslocar-se em frente, desencadeia o comportamento STOP e pára imediatamente (ou quase, devido ao atraso associado à travagem).

No GUI, onde são recebidos no PC os dados provenientes do *robot*, são representados os dados sensoriais à medida que vão sendo recebidos (note-se que são representados apenas os dados dos sensores, e não a posição do *robot*); estes dados chegam com uma periodicidade de 500ms. Se se considerar este atraso desprezável, o que parece sensato tendo em conta a baixa velocidade de deslocamento do *robot*, temos uma representação da informação em tempo-real.

Finalmente, o *software* de navegação utiliza os dados sensoriais recolhidos pelo GUI. Na implementação actual, este apenas guarda os dados quando o teste está concluído e é terminada a comunicação com o veículo robótico, pelo que a noção de tempo-real aqui não existe. A representação da posição do *robot* no mapa é feita "*offline*", quando toda a acção já ocorreu.

No entanto, isto apenas foi feito assim para facilitar o processo da aquisição de dados, uma vez que a implementação do software de navegação no PC não faz parte dos objectivos deste trabalho. O processamento e acondicionamento dos dados no *Arduino* é feito de forma a minimizar os atrasos associados às trocas de informação. Pode-se considerar o envio de dados como um processo em tempo-real. Da mesma forma, o algoritmo de navegação está preparado para processar dados à medida que lhe vão sendo fornecidos, e tem uma elevada capacidade de processamento. Portanto o sistema, quer do

lado do *Arduino* quer do lado do PC, está preparado para funcionar em tempo real, e é esse o verdadeiro objectivo da navegação neste projecto.

Resumindo, tempo-real existe no *Arduino*, e pode também considerar-se ao nível da representação dos dados no GUI. Presentemente, a representação da posição do veículo no *software* de navegação é feita sem ser em tempo-real. Este facto não constitui uma limitação significativa, pois a representação em tempo-real neste cenário passa simplesmente por organizar a recepção dos dados sensoriais de forma mais adequada e garantir que o sistema que executa o algoritmo de navegação tem capacidade computacional suficiente para efectuar os cálculos de navegação em tempo-real.

## 5.9. Integração no Sistema de Navegação Cooperativa (PC)

O presente trabalho, apesar de não incluir testes de navegação cooperativa, disponibiliza as componentes fundamentais de *software* e *hardware* para futuras implementações daquele tipo de navegação. Estas componentes são:

- Comunicações – diferentes tipos de comunicação e topologias já referidas; ver Capítulo 2;
- Controlo de kits robóticos – PID, comportamentos, etc.
- Aquisição de dados sensoriais e respectiva temporização optimizada para comunicação com o PC;
- Navegação centralizada no PC – permite estimar a localização de múltiplos veículos visando o elevado poder computacional do PC e responsabilizando este pelo comando dos diferentes veículos. Notar que o nó que usa o PC como sistema de processamento pode corresponder a um dos veículos funcionando como "supervisor", desde que este disponha de poder computacional suficiente (usando por exemplo uma placa do tipo *Raspberry Pi*).

# 6. Resultados da Navegação

## 6.1. Descrição do teste e cenário de teste

Neste teste final, pretendeu-se avaliar o funcionamento global do sistema, integrando os dados recolhidos pelo veículo robótico com o algoritmo de navegação e comparar o trajecto real com o estimado pelo filtro de partículas no PC.

O teste realizado aqui passa por utilizar os dados sensoriais adquiridos durante um percurso no corredor representado na Figura A.1, tais como os apresentados no capítulo anterior, na secção **5.6**. Adicionalmente, aqui foram registadas com base em observação visual algumas posições reais do veículo robótico no corredor, para poder desenhar razoavelmente o seu trajecto. Com isto, o objectivo será comparar três trajectórias no mesmo mapa:

- Trajectória real;
- Trajectória estimada com base na odometria;
- Trajectória estimada pelo filtro de partículas, que utiliza os dados dos sensores de distância, *encoders*, e nalguns pontos também os dados da bússola.

Aqui pretende-se avaliar o desempenho do algoritmo de navegação, quando este utiliza os dados sensoriais fornecidos. Neste teste utilizou-se uma implementação do controlo dos motores, que corresponde à **Segunda Abordagem de controlo** mencionada na secção **5.6.1.**

É ainda importante referir que neste teste, fez-se questão de fechar o *loop* no percurso, isto é, antes de terminar o percurso, o *robot* voltou a passar pela posição inicial (real).

Note-se que a recolha dos dados é feita em tempo real, mas a reconstrução da trajectória por parte do algoritmo de navegação é feita *offline*, isto é, depois de o veículo robótico ter executado todo o percurso. No entanto, todos os dados sensoriais utilizados são reais, devidamente temporizados e, uma vez que as comunicações asseguram o envio dos dados em tempo real, a navegação poderia ter sido feita também em tempo real.

O cenário escolhido para executar este teste é o ilustrado na Figura A.2. Foi escolhido por ser mais interessante do ponto de vista da simetria do corredor, pois neste cenário duas das portas foram ocultadas e isso ajuda o *robot* a resolver algumas situações de localização ambígua. O percurso no corredor é o mesmo que está ilustrado na **Figura 5.11.**

## 6.2. Resultados

Os dados sensoriais resultantes da aquisição durante o percurso são apresentados nos gráficos das figuras abaixo.

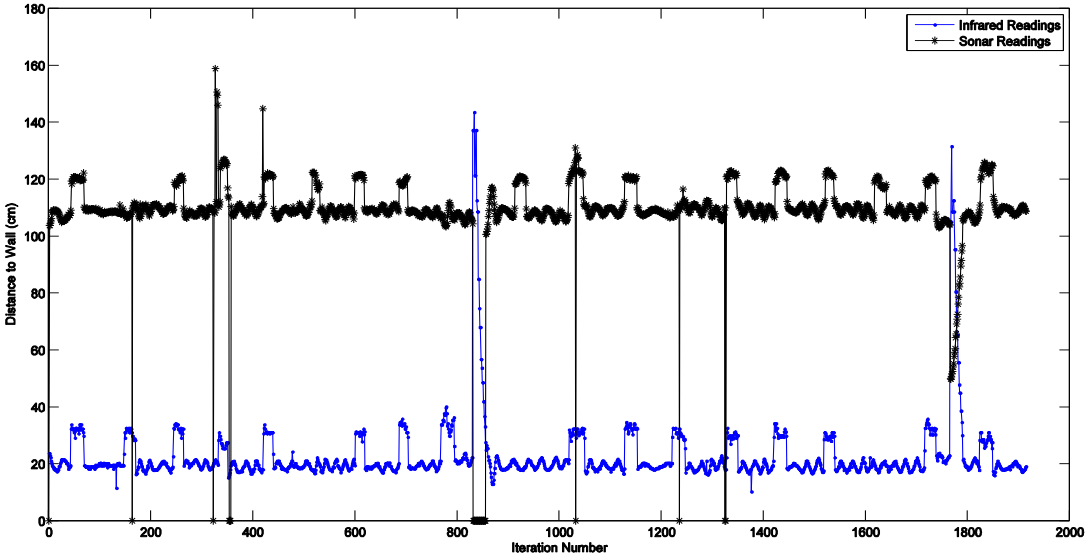



**Figura 6.1** – Dados adquiridos pelos sensores de distância ao longo do percurso.

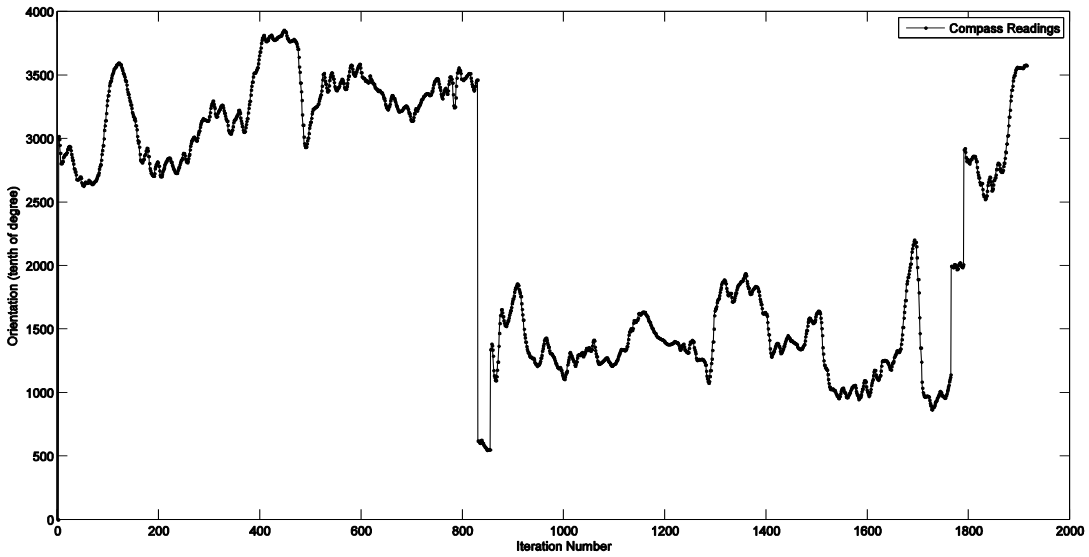



**Figura 6.2** – Dados adquiridos pela bússola ao longo do percurso.

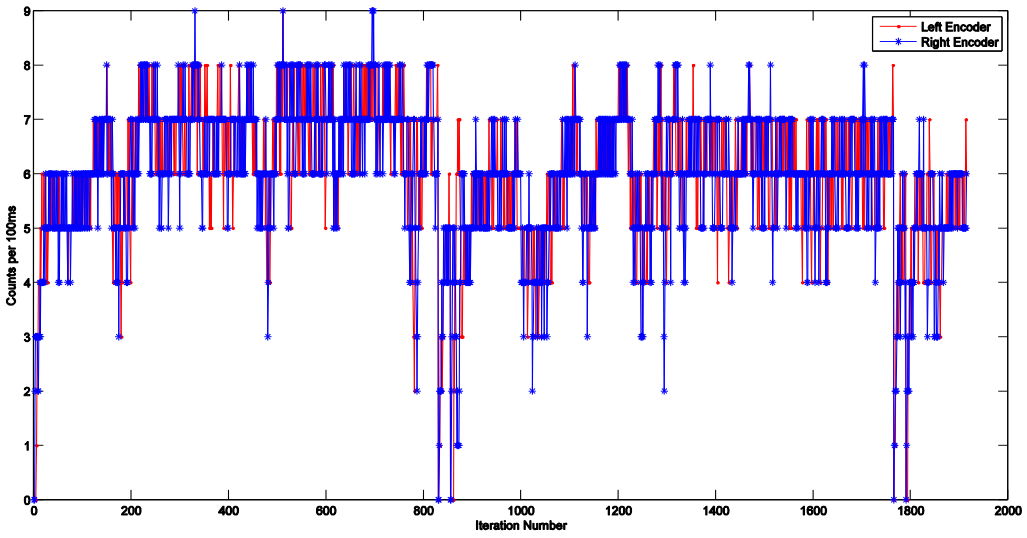



**Figura 6.3** – Dados adquiridos pelos *encoders* ao longo do percurso.

Como se pode observar, os resultados são muito semelhantes aos adquiridos na secção **5.6**. No entanto, note-se que aqui os dados não estão representados no tempo, pois devido à necessidade de paragem para proceder ao registo da posição real do *robot* na transição entre estados, o gráfico temporal tem muitos tempos "mortos" e torna-se pouco claro.

No gráfico dos dados dos sensores de distância (**Figura 6.1**) é fácil de identificar as portas de um lado e de outro, com o infravermelhos a detectar a parede mais próxima e o sonar a mais longínqua. Nota-se ainda, observando os dados do sensor infravermelhos, os pontos de viragem após os quais o veículo se vai aproximando de um obstáculo (a parede, neste caso). Estes instantes correspondem aproximadamente às medidas 830 e 1770.

Os dados do sonar têm algumas medidas com erro coincidentes com os pontos em que existem esquinas das reentrâncias das portas. Tal pode ser explicado devido ao *multipath* do feixe emitido pelo sensor, que ao embater na esquina, reflecte noutra direcção que não a do receptor do sensor, e portanto chega mais tarde – o que é interpretado como uma medição maior que a real – ou não chega sequer dentro do tempo limite definido no dispositivo – dando as medições de zero.

Estes dados diferem dos obtidos no capítulo anterior (secção **5.6**) principalmente pelo facto de aqui o percurso ser ligeiramente mais comprido, devido ao fecho do *loop*.

Relativamente aos dados provenientes da bússola digital, é notória a frequente existência de anomalias magnéticas. Isto mostra que apenas com a bússola utilizada, não seria possível o seguimento de uma trajectória rectilínea com base em leituras da orientação. No entanto, tal não é problemático, pois estes dados são usados no algoritmo de navegação apenas para detectar as rotações de grande magnitude do *robot* ao longo do trajecto. As rotações correspondem a variações bruscas na orientação em medidas consecutivas da bússola, tal como acontece perto das medidas 830 ou 1770 (ver **Figura 6.2**).

Acerca dos *encoders*, pode-se referir que as suas contagens de rotações estão praticamente em concordância (**Figura 6.3**), o que significa que na ausência de perturbações

ao nível mecânico (derrapagens ou desigualdade nos diâmetros das rodas) a trajectória é aproximadamente rectilínea.

A Figura 6.4 apresenta as trajectórias resultantes da execução do algoritmo de navegação. Após algum tratamento dos dados, como a filtragem de dados incorrectos provenientes do sonar, o algoritmo de navegação processa a informação disponível e estima as duas trajectórias desejadas: com base na odometria, e com base nos dados sensoriais disponíveis. Recorrendo às localizações reais do robot registadas aquando da aquisição no percurso, traçou-se ainda no mesmo gráfico a trajectória real (aproximada).

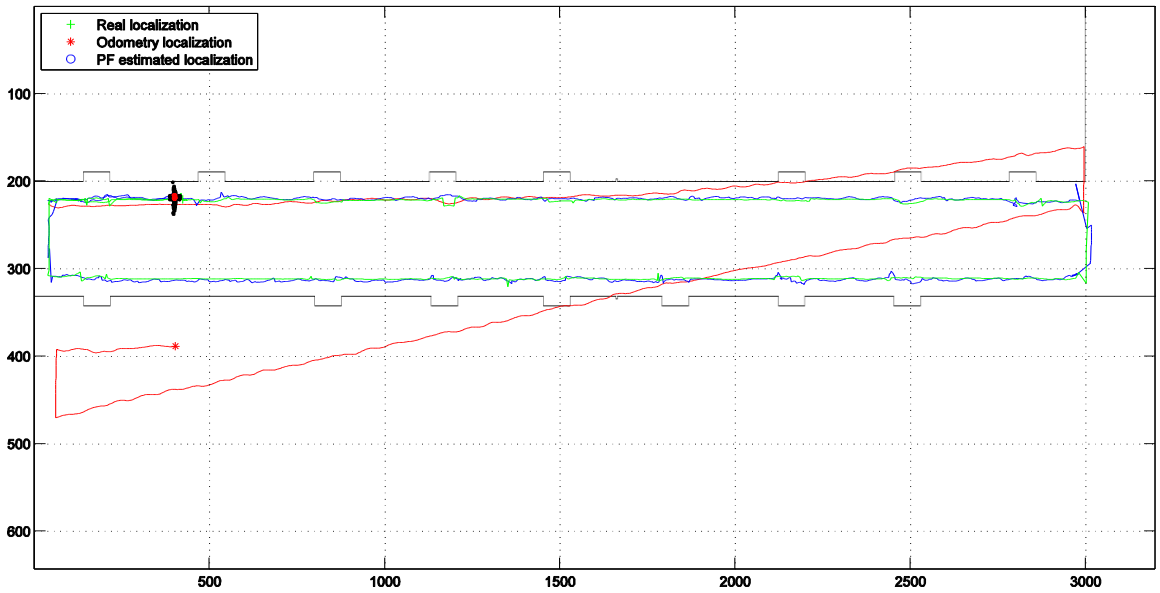



**Figura 6.4** – Trajectórias resultantes do teste de navegação: real, estimada pela odometria, e estimada pelo filtro de partículas (ambos os eixos em centímetros).

Na **Figura 6.4**, observam-se principalmente quatro factores interessantes: as três trajectórias esperadas, e a dispersão de partículas no final do percurso. Esta dispersão é semelhante à que é muitas vezes referida na literatura, como por exemplo em **[30]**. Esta distribuição das partículas deve-se essencialmente ao erro associado à odometria, pois o deslocamento medido aproxima-se muito do real, mas este nem sempre é em linha recta, como era suposto. Tal implica que no fim do percurso, a incerteza na localização é muito superior em termos da orientação associada ao movimento do que na própria translação, pelo que as partículas se distribuem segundo um arco transversal ao sentido do deslocamento.

Está bem realçado o erro na odometria (trajectória a vermelho). Em termos da distância percorrida, a trajectória calculada com base na odometria aproxima-se da trajectória real; no entanto, devido a erros sistemáticos na medição da direcção do deslocamento, desvia-se muito da trajectória real (ao fim do corredor, a trajectória com base na odometria está cerca de 60cm ao lado da trajectória real). Apesar de ser ainda elevado, este erro foi atenuado através de uma calibração da odometria baseada na comparação entre os pontos iniciais e finais da trajectória real com os pontos respectivos da trajectória obtidos com base na odometria. É ainda interessante destacar que, apesar do erro associado à trajectória da odometria, na forma desta é facilmente identificável o formato do percurso

no corredor. Ao fim do percurso, incluindo o fecho do *loop*, o erro na distância euclidiana entre a posição real e a estimada pela odometria é de cerca de 170cm.

Quanto à trajectória estimada pelo filtro de partículas (a azul), é notória a coincidência com a trajectória real (a verde). A trajectória estimada tem alguns desvios, influenciada pelos erros associados às medidas de distância provenientes dos sensores e também pelo erro da odometria, já analisado. Porém, a localização ao longo do percurso é muito bem conseguida, apresentando erros da ordem dos centímetros, ao longo de um corredor de 30m com cerca de 1,5m de largura. No fim o percurso, a diferença entre a trajectória real e a estima pelo filtro de partículas é aproximadamente 25cm.

Os pontos onde a localização foi mais pobremente conseguida foram os pontos de rotação, pois durante as rotações, o *robot* não faz aquisição sensorial, e os sensores deslocam-se (em relação ao referencial global) mais do que o próprio veículo. Estes "buracos" na aquisição não são desprezáveis, pois durante as rotações os sensores movem-se das duas posições cerca de 20cm. No entanto, Nota-se, pelo trajecto a azul, que o filtro converge rapidamente para o trajecto verde (real).

É ainda interessante analisar três situações que surgem da estimação das trajectórias. As figuras seguintes ilustram: a situação inicial, uma situação de pouca incerteza na localização, e finalmente uma de elevada incerteza na localização.

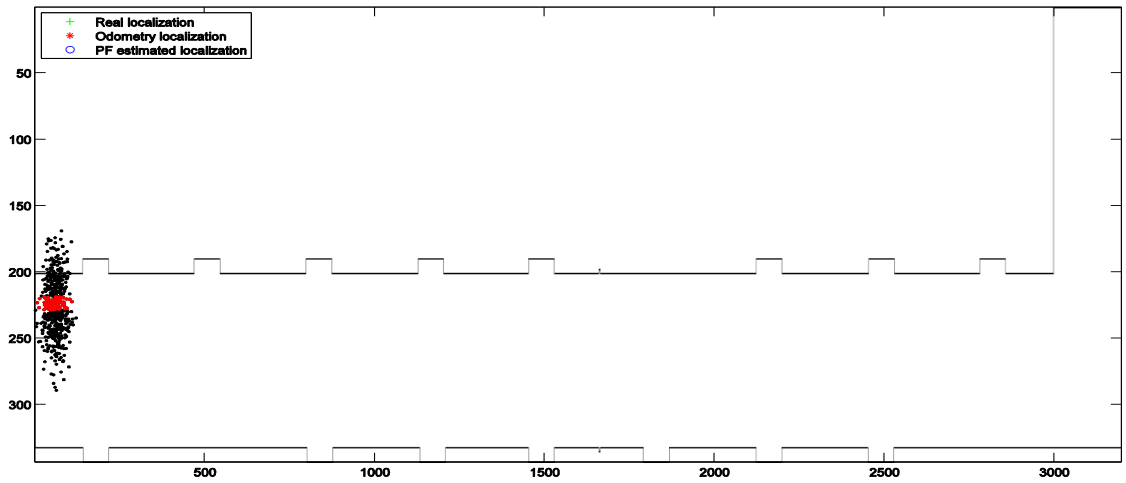



**Figura 6.5** – Distribuição inicial das partículas pelo espaço (ambos os eixos em centímetros).

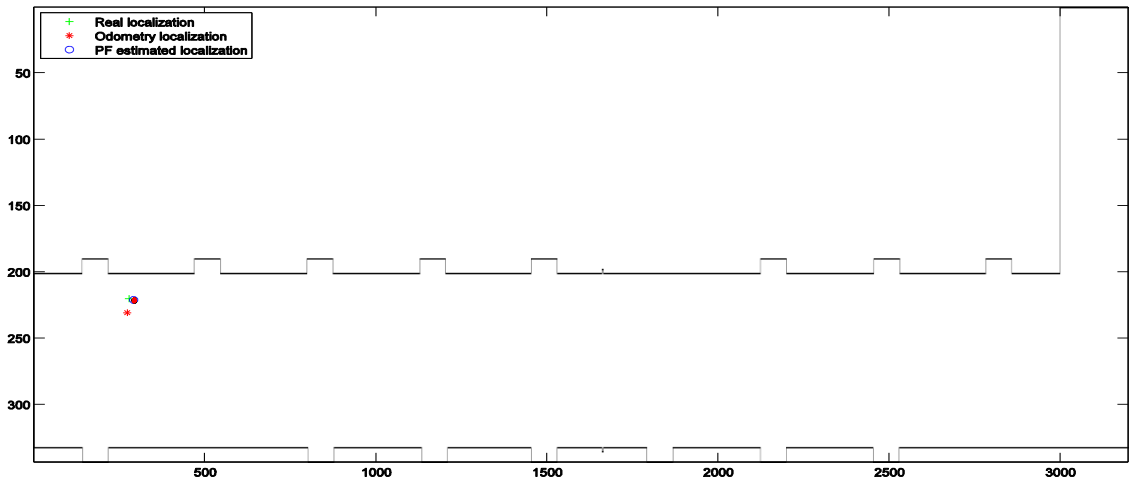



**Figura 6.6** – Distribuição das partículas, após algumas iterações (ambos os eixos em centímetros).

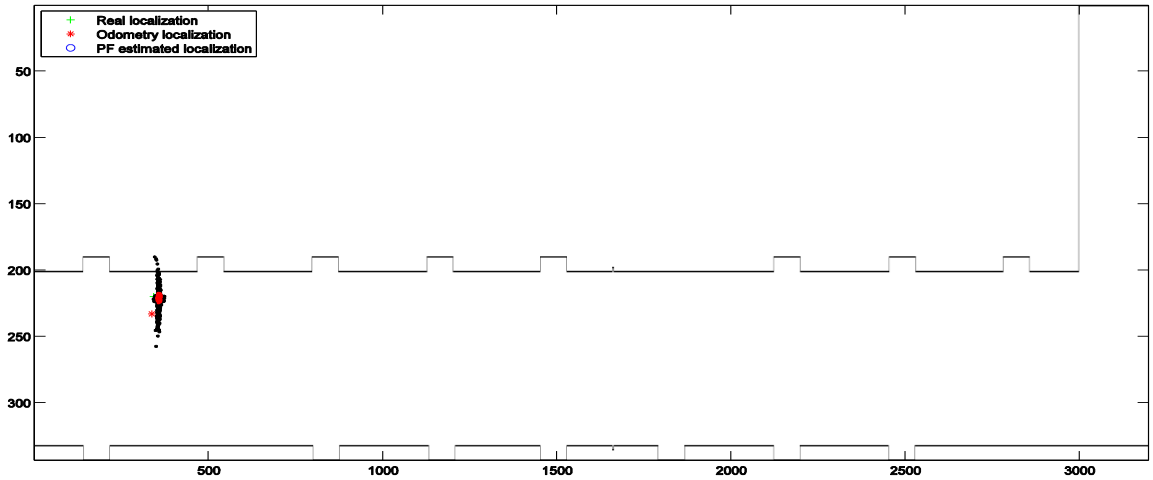



**Figura 6.7** – Distribuição das partículas num instante posterior ao da **Figura 6.6** (ambos os eixos em centímetros).

A **Figura 6.5** ilustra a distribuição das partículas no instante inicial, em que são distribuídas no espaço em torno da posição inicial (incerta). Note-se que a representação espacial das partículas em cada iteração segue uma função densidade de probabilidades que é uma mistura de duas distribuições de probabilidades: uma uniforme e outra calculada pelo filtro de partículas. Na primeira iteração, esta mistura inclui uma distribuição normal e uma uniforme.

Na **Figura 6.6** o veículo robótico já se deslocou, tendo passado por uma porta. Aqui o erro acumulado da odometria ainda é reduzido, e a passagem por reentrâncias ajuda na localização do veículo (no eixo dos *xx*). Estas circunstâncias justificam a incerteza praticamente desprezável na localização do *robot*. As partículas estão muito concentradas e são praticamente coincidentes.

À medida que o veículo se desloca ao longo da parede, na posição ilustrada na **Figura 6.7** ainda não passou por outra reentrância, e o erro associado à odometria vai aumentando. A juntar a isso, os sensores de distância também vão introduzindo erro nas estimações. Estes factores contribuem para que a incerteza da localização aumente, e as partículas se dispersem (principalmente no eixo dos *yy*).

Estes cenários ilustram muito bem o que acontece tipicamente na localização de robots em ambientes deste tipo. À medida que o veículo robótico avança, a incerteza vai aumentado e as partículas vão-se dispersando pelo espaço. Quando o robot passa por ponto de referência (*landmark*), utiliza-o na estimação da localização e reduz a incerteza, concentrando mais as partículas.

## 6.3. Potencial da Navegação Magnética

Os testes executados neste trabalho, quer no contexto do controlo, quer no contexto de teste à navegação, foram executados no mesmo ambiente, seguindo o mesmo percurso. Em todos eles foram obtidos e apresentados os dados provenientes dos diferentes sensores no veículo robótico.

Apesar de estes testes terem sido executados em dias diferentes e com desenvolvimentos diferentes, os dados sensoriais adquiridos pela bússola digital são muito semelhantes. Esta semelhança pode ser observada nos resultados apresentados neste capítulo e nos obtidos no capítulo anterior, na secção **5.6**. Este resultado é muito importante, pois mostra que a assinatura magnética de um dado ambiente (*indoor*, como é o caso) pode ser utilizada para fazer navegação. Isto potencia ainda mais a fusão sensorial de sensores básicos (baratos) aplicada em robótica móvel cooperativa.

O tema da navegação robótica com base na assinatura magnética do ambiente envolvente é bastante recente, havendo actualmente uma grande aposta em termos de investigação, neste tipo de soluções.

Em **[42]** é feito um levantamento e estudo da assinatura magnética de um ambiente *indoor*. Em **[43]** é usada a assinatura magnética em quatro edifícios para localização global, utilizando um filtro de Monte Carlo.

## 6.4. Comentários Finais

Os resultados da navegação são muito satisfatórios e motivantes para o constante melhoramento do trabalho, pois o objectivo era construir um veículo robótico o mais simples e barato possível capaz de se deslocar autonomamente e de enviar dados sensoriais para serem utilizados em navegação. Este objectivo foi conseguido, deixando ainda margem de evolução. A partir daqui, a navegação cooperativa depende apenas de o PC ter capacidade de processamento suficiente para executar o algoritmo de navegação para localização de mais de um veículo simultaneamente.

É importante salientar que o erro de odometria, mesmo calibrado, é muito elevado e que mesmo assim, usando os dados sensoriais de distância, é possível executar a navegação com elevada precisão.

Considera-se portanto que é viável executar este tipo de navegação usando *hardware* de baixo custo.

# 7. Conclusões e Trabalho Futuro

## 7.1. Conclusões

Finda a descrição do trabalho, existem diversas ilações a serem tiradas.

A quantidade de sensores (e motores) que existem é muito vasta. Existem sensores com as mais diversas características, que se adequam às diferentes aplicações. Ao nível do processamento também se verifica esta situação: existem várias alternativas ao *Arduino*, surgindo actualmente cada vez mais projectos que recorrem ao *Raspberry Pi* como unidade de processamento. Também ao nível das comunicações existem outras soluções alternativas ao *ZigBee*.

Neste trabalho, o princípio subjacente à escolha de componentes para a construção do veículo robótico escolher o mais económico do mercado e que ainda assim garantisse um bom nível de desempenho. Os elementos da estrutura do veículo foram implementados com material reciclado; por exemplo, o chassis do *robot* é o suporte para papel de uma antiga impressora.

O objectivo do trabalho era desenvolver um sistema robótico capaz de executar tarefas simples, e que enviasse os dados sensoriais adquiridos para uma unidade central de processamento, um PC neste caso. Pretendia-se também que o veículo robótico recebesse dados provenientes do PC, como por exemplo comandos para transição de estado ou para executar determinada tarefa.

Como foi demonstrado ao longo deste documento, foi possível cumprir satisfatoriamente todos os objectivos. Além de os resultados serem bastante entusiasmantes, existe bastante margem para melhorar o sistema, quer ao nível do veículo robótico, quer ao nível do PC.

Um aspecto relevante de analisar é o das comunicações. Como foi relatado desde o início, a comunicação *wireless* entre o veículo e o PC é a chave para o funcionamento do conceito essencial do projecto: o *robot* fazer praticamente apenas a aquisição sensorial, deixando o processamento pesado para o PC, sendo este o responsável por decidir os principais comportamentos a serem tomados pelo veículo. A tecnologia *ZigBee*, bem como os rádios *XBee* utilizados, cumpriram os requisitos das comunicações, assegurando a troca de pacotes entre os nós da rede em condições de serem processados em tempo real. Quando a comunicação se tornou pesada, atrasando o processamento, encontraram-se soluções para contornar o problema, como empacotamento de dados de forma a reduzir a quantidade de pacotes trocados.

Os sensores de baixo custo utilizados apresentam erros de maior intensidade. No entanto, recorrendo a técnicas de processamento de sinal, ou até mesmo alterações simples dos sensores (no caso do sonar *range finder*, ver subsecção **3.7.2**), os erros foram bastante atenuados.

No capítulo **6**, os dados sensoriais adquiridos no percurso do veículo robótico ao longo do corredor do Departamento de Geociências (ver Figura A.1), foram utilizados no algoritmo de navegação. Como foi analisado no referido capítulo, os resultados foram bastante animadores, pois com sensores e actuadores com ruído significativo, nomeadamente ao nível da odometria, o trajecto estimado pelo filtro de partículas aproxima-se bastante da trajectória real percorrida pelo *robot*.

Nesta dissertação conseguiu-se a interligação entre o sistema robótico desenvolvido, que está preparado para fornecer dados sensoriais a um PC, e um algoritmo de navegação já existente, que estava preparado para processar dados sensoriais para estimação da localização de um kit robótico num mapa previamente fornecido.

Este trabalho mostra que os conceitos aplicados funcionam; os resultados comprovam-no e são promissores. O *ZigBee* é uma boa aposta para aplicações de robótica móvel em ambientes *indoor*. Os sensores usados são suficientemente precisos para a aplicação da navegação. Os filtros de partículas de Monte Carlo são uma solução robusta e fiável para algoritmos de navegação.

## 7.2. Trabalho Futuro

A título de melhorias futuras no trabalho podem indicar-se as seguintes, transversalmente aos componentes do sistema desenvolvido.

Ao nível do *hardware* do veículo robótico, algumas melhorias a fazer futuramente são:

- Acrescentar (ou simplesmente trocar a existente) uma bússola com compensação de inclinação, visto que a actual é muito propensa a erros caso não esteja horizontal;
- Utilização de dois sensores de distância sonar sincronizados, de forma a reduzir o erro introduzido pelos lobos laterais e abertura do feixe;
- Introdução de um detector mecânico de colisão.

No PC, existem também diversas alterações para melhorar o sistema, tais como:

- Ao nível do GUI, acrescentar alguns comandos práticos, como um botão de paragem, ou de velocidade pré definida;

Ao nível do *software* implementado no *robot*, poderão melhorar-se os seguintes aspectos:

- Aumentar a resolução dos *encoders*;
- Implementar mais comportamentos, nomeadamente evitar obstáculos;

- Para além do envio dos dados sensoriais, enviar também via rede *wireless* para o PC o estado actual do veículo, cada vez que ocorrer uma transição, de forma a aumentar a eficácia do *software* de navegação executado no PC.

# Anexo A. Mapas

As figuras seguintes representam os cenários construídos no corredor descrito em **4.4**. A Figura A.2 corresponde ao mesmo corredor, com duas portas ocultadas.

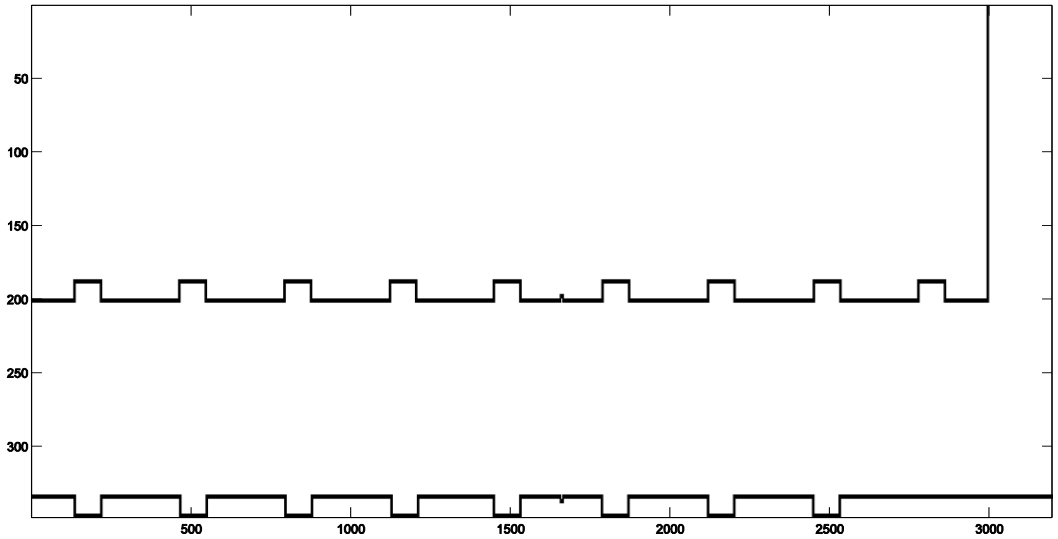



**Figura A.1** – Mapa do corredor do Departamento de Geociências, onde foram feitos os testes ao controlo de trajectórias e de navegação.

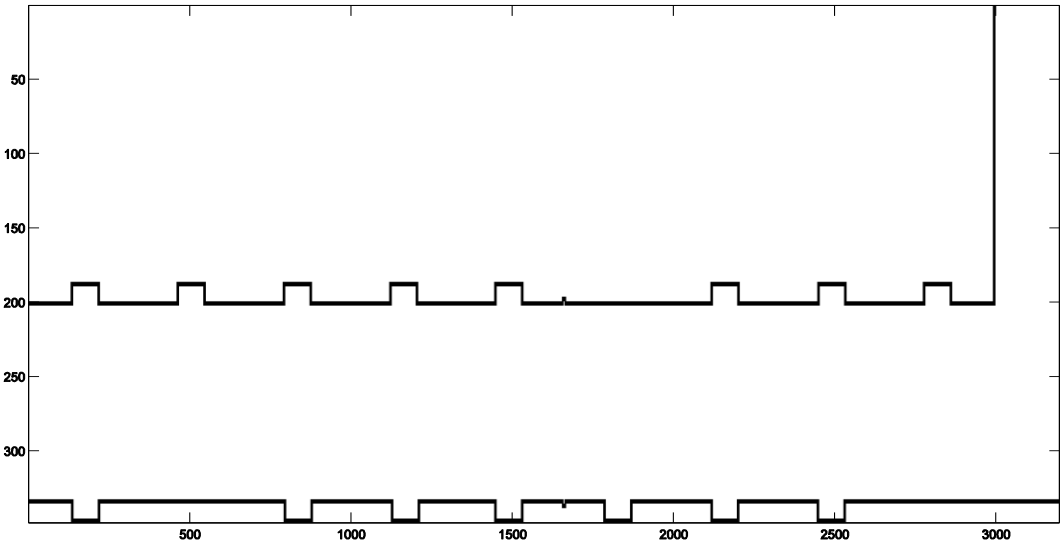



**Figura A.2** – Cenário onde foi feita a aquisição de dados por parte do veículo robótico para teste da navegação.

# Anexo B. Interface Gráfico

Na figura seguinte é apresentado o GUI, com a legenda e função de cada item numerado.

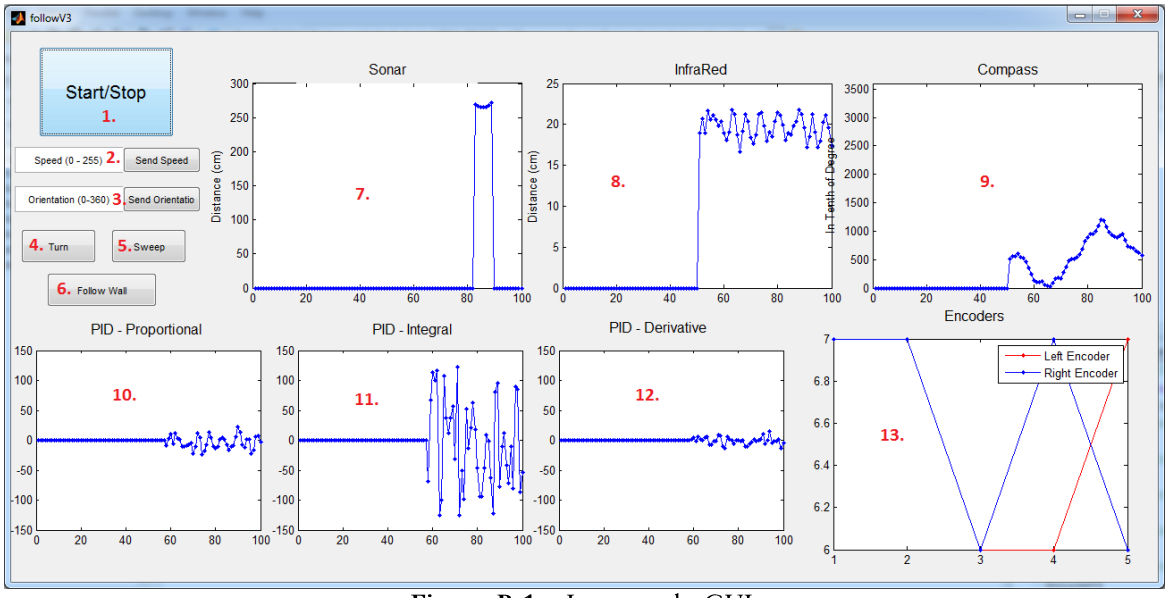



**Figura B.1** – Imagem do GUI.

1. ***Start/Stop*** – Botão para iniciar/terminar a comunicação com o veículo robótico (sem enviar qualquer comando).
2. ***Send Speed*** – Caixa de texto para indicação da velocidade pretendida e respectivo botão para enviar a instrução.
3. ***Send Orientation*** – Caixa de texto para indicação da orientação angular absoluta (em relação ao norte magnético), e respectivo botão para envio do comando.
4. ***Turn*** – Botão para envio de instrução para o *robot* efectuar uma rotação (relativa) de 90° à direita. Futuramente este comando será genérico, isto é, será passado como argumento o sentido e ângulo de rotação juntamente com o comando.
5. ***Sweep*** – Botão para envio da instrução para o veículo robótico fazer um varrimento de 180° com os sensores de distância. Em trabalho futuro, será enviado como argumento a abertura angular do varrimento sensorial.
6. ***Follow Wall*** – Botão de envio do comando para o veículo seguir paralelamente a um obstáculo (tipicamente uma parede) à sua esquerda.
7. ***Sonar*** – Área onde vão sendo representados as medições de distância provenientes do sonar à medida que são recebidos pelo PC.
8. ***InfraRed*** - Gráfico onde vão sendo apresentados as medições de distância provenientes do sensor infravermelhos à medida que são recebidos pelo PC.
9. ***Compass*** – Espaço onde vão sendo apresentados as medições de orientação provenientes da bússola digital à medida que são recebidos pelo PC.

10. ***PID–Proportional*** – Área onde vai sendo apresentado o erro da componente proporcional do PID, calculado no *robot*.
11. ***PID-Integral*** – Espaço onde vai sendo representado o erro da componente integral do PID, calculado no *robot*.
12. ***PID-Derivative*** - Área onde vai sendo representado o erro da componente derivativa do PID, calculado no *robot*.
13. ***Encoders*** – Gráfico onde vão sendo apresentadas as últimas contagens dos *encoders*, à medida que vão chegando ao PC.

# Anexo C. Esquema Eléctrico

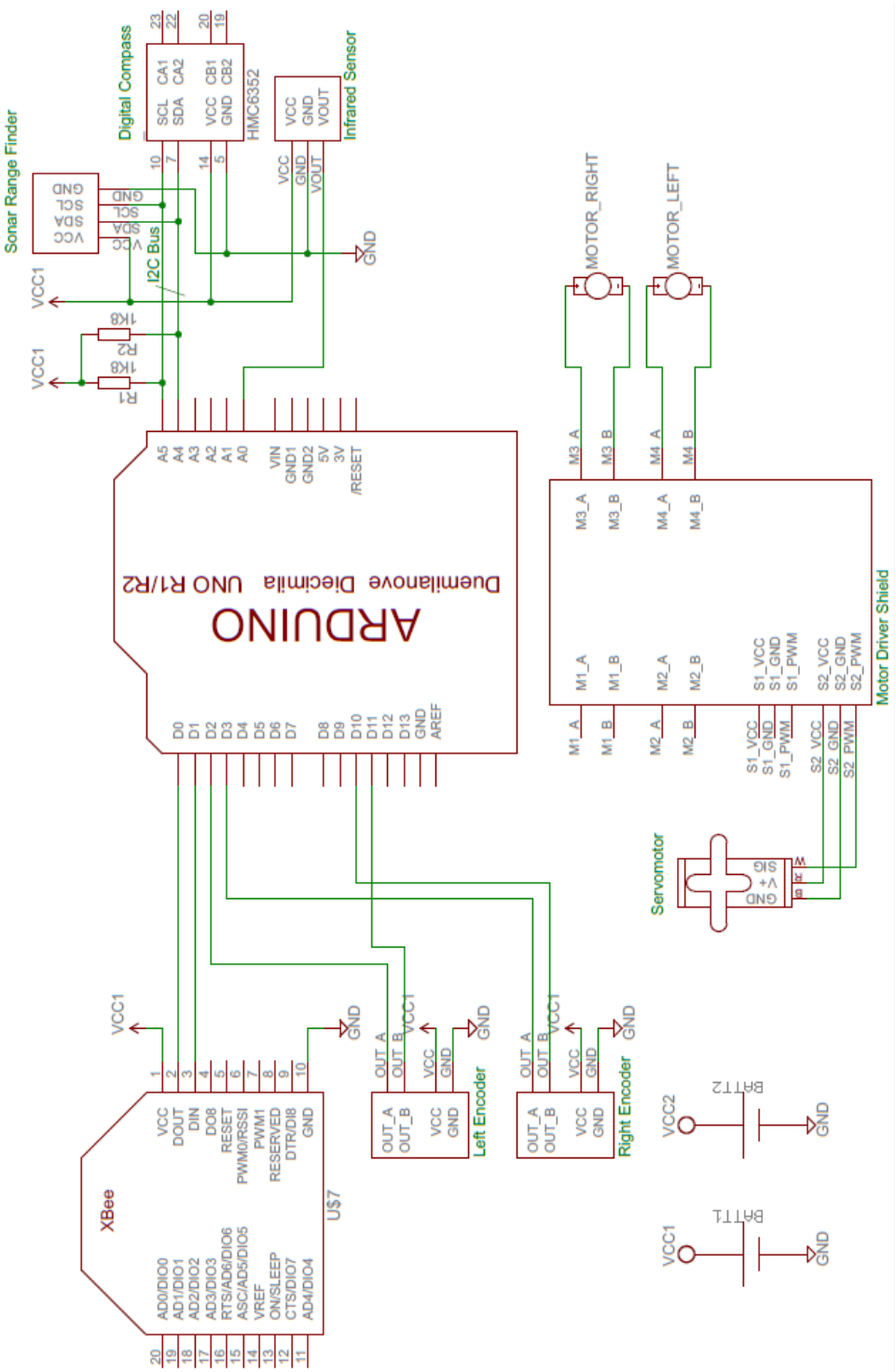



**Figura C.1** – Esquema eléctrico do veículo robótico. Note-se que existe um encaixe entre o *shield* e o *Arduino*, havendo uma correspondência de pino a pino entre D5 e M3_A, D6 e M4_A, D9 e S2_PWM, conforme está descrito na **Tabela 5.2.**

# Anexo D. Conteúdo do CD

No CD que se encontra em apêndice a este documento encontra-se todo o material envolvido neste trabalho.

São incluídos documentos de teoria da matéria estudada, notas de fabricantes do *hardware*, textos de apoio (tutoriais), bem como rotinas desenvolvidas para *Arduino* e *Matlab* para testes de calibração e também para testes mais complexos, como os de navegação. Constam também ficheiros com resultados obtidos, gráficos e também fotos e vídeos do *robot* em execução de tarefas.

Na tabela seguinte são apresentadas as pastas mais relevantes e a descrição dos documentos/ficheiros contidos.

**Tabela D.1** – Conteúdo do CD.

| Pasta | Descrição | |
|---|---|---|
| Bússola Digital | Documentação e rotinas de testes, e seus resultados. | |
| Caracterização Sensores de Distância | Dados de aquisição sistemática para caracterização dos sensores IR e US. | |
| Cinemática | Documentação teórica sobre a cinemática em robótica móvel. | |
| Escrita | Documentos de texto da presente dissertação. | |
| Fotos | Fotos e vídeos do veículo robótico e seus componentes. | |
| IR Sensor GP2D120 | Documentação e rotinas de testes ao sensor IR utilizado. | |
| Mapas | Mapas do corredor em ficheiros *Matlab* e em imagens. | |
| Motores | Documentação sobre motores, *encoders* e *shield* de motores utilizados; rotinas de teste do funcionamento. | |
| PID | Rotinas para afinação dos parâmetros do PID e seus resultados gráficos. | |
| Programação por Comportamentos | *behaviorProgramming_v3* | Código final para *Arduino*, utilizado nos testes de navegação e controlo. |
| | *followV3.m* | Código *Matlab* final para interacção com o *robot*; utilizado nos testes de navegação e controlo. |
| | *descrição.txt* | Descrição do conteúdo de cada rotina implementada em *Arduino* e *Matlab*. |
| | *resultados.txt* | Descrição do conteúdo de cada ficheiro de resultados (dos testes de controlo e navegação). |
| Resultados | Figuras resultantes da execução do algoritmo de navegação sobre os dados obtidos pelo robot. | |
| Servo | Rotina de teste do funcionamento do servo utilizado. | |
| Sonar SRF10 | Documentação do fabricante e para melhorias ao sensor (subsecção 3.7); rotinas de teste do funcionamento. | |
| Sweep Sensores Distância | Rotinas e resultados de varrimento para aquisição com sensores de distância IR e US. | |
| ZigBee | Documentação teórica acerca do *ZigBee*, nota do fabricante dos módulos *XBee*, integração com *Arduino*, e rotinas de teste. | |

# Referências


[1] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi," *The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2007.

[2] P. Mirowski, R. Palaniappan, and T. K. Ho, "Depth camera SLAM on a low-cost WiFi mapping robot," *Technologies for Practical Robot Applications (TePRA), 2012* 2012.

[3] A. Gil-Pinto, P. Fraisse, and R. Zapata, "Wireless Reception Signal Strength for Relative Positioning in a Vehicle Robot Formation," *Robotics Symposium, 2006. LARS '06.*, 2006.

[4] A. S. L. Fernandes, "Comunicação Ad Hoc em Equipas de Robôs Móveis Utilizando a Tecnologia ZigBee," Mestre, Departamento de Engenharia Eletrotécnica, Universidade de Coimbra, 2012.

[5] R. G. Shepherd and S. a. P. Mansoor, "Bluetooth Based Proximity Sensing for Reactive Mobile Robots," *TENCON 2005 2005 IEEE Region 10*, 2005.

[6] G. Mester, "Wireless Sensor-based Control of Mobile Robots Motion," *Intelligent Systems and Informatics, 2009. SISY '09*, 2009.

[7] IEEE. (2006). *IEEE 802.15 WPAN™ Task Group 4 (TG4).* Available: http://www.ieee802.org/15/pub/TG4.html

[8] B. Zhang and S. Gao, "The Study of Zigbee Technology's Application in Swarm Robotics System," *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011*, 2011.

[9] C.-H. Lin and K.-T. Song, "Location Estimation Using Probability Map of a ZigBee Sensor Network," *Advanced Robotics and Intelligent Systems (ARIS), 2013*, 2013.

[10] N. Instruments, "Title," unpublishedl.

[11] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*: John Wiley & Sons, 2010.

[12] S. Shue and J. M. Conrad, "A Survey of Robotic Applications in Wireless Sensor Networks," *Southeastcon, 2013 Proceedings of IEEE*, p. 5, 2013.

[13] Z. Alliance. *ZigBee.* Available: www.zigbee.org

[14] S. Safaric and K. Malaric, "ZigBee wireless standard," *Multimedia Signal Processing and Communications*, 2006.

[15] D. International, "XBee®/XBee-PRO® ZB RF Modules," ed, 2012.

[16] J. L. Jones, A. M. Flynn, and B. A. Seiger, *Mobile Robots. Inspiration to Implementation*, 2nd ed.: A K Peters, 1999.

[17] J. D. Madden. (2007). *Mobile Robots: Motor Challenges and Materials Solutions.* Available: http://www.sciencemag.org/content/318/5853/1094.short

[18] M. D. Adams, *Sensor Modelling, Design and Data Processing for Autonomous Navigation* vol. 13: World Scientific, 1999.

[19] R. Kuc and M. W. Siegel, "Physically Based Simulation Model for Acoustic Sensor Robot Navigation," *Pattern Analysis and Machine Intelligence*, 1987.

[20] A. P. Aguiar, "Modelização, Controlo e Condução de um veículo submarino autónomo para o transporte de laboratórios bênticos," MSc, DEEC, Instituto Superior Técnico, 1996.

[21] A. Robotics. (2006). *Linearizing Sharp Ranger Data.* Available: http://www.acroname.com/robotics/info/articles/irlinear/irlinear.html

[22] J. Borenstein, H. R. Everett, and L. Feng, *Navigating Mobile Robots*: A K Peters, 1996.

[23] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*: Cambridge University Press, 2000.

[24] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *Robotics and Automation*, 1992.
[25] R. Arkin, *Behavior-Based Robotics*: MIT Press, 1998.
[26] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*: Prentice Hall, 2002.
[27] M. I. Ribeiro, "Uma Viagem ao Mundo dos Robots," ed. Ciclo de Colóquios "Despertar para a Ciência 2004", 2004.
[28] E. Wise, *Applied Robotics*: Prompt Publucations, 1999.
[29] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*: Springer, 2008.
[30] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*: MIT Press, 2005.
[31] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, 2002.
[32] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, *et al.*, "Particle filters for positioning, navigation, and tracking," *IEEE Transactions on Signal Processing*, 2002.
[33] J. Engelberger, "Economics should be main basis for using robots," *Production Engineer*, 1983.
[34] NASA. (2013). *Mars - Exploration Rovers.* Available: http://marsrovers.jpl.nasa.gov/home/
[35] F. Ducatelle, G. A. Di Caro, C. Pinciroli, F. Mondada, and L. Gambardella, "Communication assisted navigation in robotic swarms: Self-organization and cooperation," *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ* 2011.
[36] M. Quigley, B. Gerkeyy, K. Conleyy, J. Fausty, T. Footey, J. Leibsz, *et al.*, "ROS: an open-source Robot Operating System," in *ICRA-2009 ???*, 2009?
[37] ROS. (2013). *ROS.org.* Available: http://wiki.ros.org/
[38] Arduino. (2013). *Arduino.* Available: www.arduino.cc
[39] R. Pi. *Raspberry Pi.* Available: http://www.raspberrypi.org/
[40] UDOO. (11-2013). *UDOO.* Available: http://www.udoo.org/
[41] H. R. Everett, *Sensors for Mobile Robots - Theory and applications*: A.K. Peters Ltd, 1995.
[42] B. Gozick, K. P. Subbu, R. Dantu, and T. Maeshiro, "Magnetic Maps for Indoor Navigation," *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, vol. 60, DECEMBER 2011 2011.
[43] J. Haverinen and A. Kemppainen, "Global indoor self-localization based on the ambient magnetic field," *Robotics and Autonomous Systems*, vol. 57, pp. 1028–1035, 31 October 2009 2009.